uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

2019-2020

Apuntes

Teoría de Automatas y Lenguajes Formales

Jorge Rodríguez Fraile[1]

[1] Universidad: 100405951@alumnos.uc3m.es | Personal: jrf1616@gmail.com

ÍNDICE GENERAL

I TEMA 1. Introducción 3

II TEMA 2. Autómatas Formales 15

III TEMA 3. Autómatas Finitos 31

IV TEMA 4. Gramáticas y Lenguajes Formales 71

V TEMA 5. Expresiones Regulares 139

VI TEMA 6. Autómatas a Pila 187

VII TEMA 7. Maquina de Turing 211

VIII TEMA 8. Complejidad Computacional 239

1. RECURSOS 249
   - 1.1. Tema 3 249
     - 1.1.1. $AFD \rightarrow AFD$ *mínimo* 249
     - 1.1.2. $AFND \rightarrow AFD$ 249
   - 1.2. Tema 4 249
     - 1.2.1. *G3 LD* → *G3 LI* 249
     - 1.2.2. Lenguaje vacío (G2) 250
     - 1.2.3. Lenguaje infinito (G2) 250
     - 1.2.4. Limpieza y bien-formación de gramáticas 250
     - 1.2.5. $G2 \rightarrow FNC$ 250
     - 1.2.6. $G2 \rightarrow FNG$ 251

1.2.7. Quitar recursividad a izquierdas 251
1.2.8. Paso de G3LD (FNG) → AF y viceversa 251
1.3. Tema 5 252
1.3.1. Teoria de síntesis 252
1.4. Tema 6 252
1.4.1. $\boldsymbol{APF \rightarrow APV}$ 252
1.4.2. $\boldsymbol{APV \rightarrow APF}$ 252
1.4.3. $\boldsymbol{G2\ (FNG) \rightarrow APV}$ 253
1.4.4. $\boldsymbol{APV \rightarrow G2}$. 253
1.4.5. Equivalencia de EERR. 254
1.4.6. Analisis 254
1.4.7. Formatos 255
1.4.8. Jeraquia de Chomsky 255

Parte I

TEMA 1. Introducción

1. Introducción a la Teoría de Autómatas y Lenguajes Formales

Grado Ingeniería Informática
Teoría de Autómatas y Lenguajes Formales

1

Objetivos

- Presentar la normativa, los contenidos y objetivos de la asignatura poniendo énfasis en las aplicaciones prácticas de la materia que se va a estudiar.
- Conocer la contextualización histórica de la Teoría de Autómatas y lenguajes formales. Desde los orígenes hasta los distintos campos de los que se ha nutrido esta área de conocimiento (Ingeniería, Lenguajes y Gramáticas, y Matemáticas y Computabilidad).
- Conocer el esquema básico que se seguirá a través de la jerarquía de Chomsky sobre los autómatas, gramáticas y lenguajes formales.
- Conocer otras máquinas abstractas relacionadas que se encuentran fuera de la jerarquía de Chomsky.
- Conocer los límites de las máquinas abstractas que se estudiarán y sus problemas de complejidad.

2

Índice

- El por qué de la Teoría de Autómatas
- Relación con otras Áreas de Conocimiento
- Máquinas, Lenguajes y Algoritmos

El por qué de la Teoría de Autómatas

Disciplinas de la Computación según la "*Educational Activities Board of IEEE*":

- Computer Engineering , **Computer Science**, Information Systems, Information Technologies, Software Engineering

Computer Science:

- "A pesar de la enorme amplitud de la informática, existen conceptos y habilidades que son comunes a la informática en su conjunto."
- "Todos los estudiantes de informática tienen que aprender a integrar la teoría y la práctica, a reconocer la importancia de la abstracción para apreciar el valor del buen diseño de ingeniería"

Fuente: Computing Curricula 2005. The Overview Report.
http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf

El por qué de la Teoría de Autómatas

- **Ciencias de la Computación**: cuerpo de conocimiento que se ocupa del estudio de los fundamentos teóricos de la información y la computación y de su implementación y aplicación en sistemas computacionales.
- Gibbs y Tucker (1986):
  - "No se debe entender que el objetivo de las Ciencias de la Computación sea la construcción de programas sino el estudio sistemático de los algoritmos y estructuras de datos, específicamente de sus propiedades formales"
    - Gibbs, N. E. and Tucker, A. B. 1986. A model curriculum for a liberal arts degree in computer science. Commun. ACM 29, 3 (Mar. 1986), 202-210. DOI= http://doi.acm.org/10.1145/5666.5667

5

El por qué de la Teoría de Autómatas

Primera inmersión en la "Teoría de la Computación":

- Es anterior al invento del Computador (incluso del transistor)
- Propiedades **MATEMÁTICAS FUNDAMENTALES** de Software, Hardware y aplicaciones de los mismos.
- Responder a preguntas como:
  - ¿Cómo puede construirse un programa para resolver un problema?
  - ¿Resuelve el programa realmente el problema?
  - ¿Cuánto se tarda en realizar un cómputo (complejidad temporal)?.
  - ¿Cuanta memoria se necesita para realizar el computo (complejidad espacial)?.
  - Y el "modelo de computación" (Imperativo, POO, Programación Lógica, etc.)
  - ¿Qué se puede computar y qué NO se puede computar?.

6

El por qué de la Teoría de Autómatas.

Aplicación directa de conceptos propios de las Ciencias de la Computación:

- Videojuegos
  - Comportamiento de personajes
- Compiladores y Procesamiento de Lenguaje Natural
  - Análisis Léxico en lenguajes programación (compilador)
  - Búsqueda de cadenas o comparación de "patrones"
  - Diseño de nuevos lenguajes de programación o ampliación
- Implementación de Protocolos Robustos
  - Para clientes o usuarios
  - E.g. Sistemas de Seguridad
- Criptografía Moderna (sus protocolos)
- …

7

El por qué de la Teoría de Autómatas.

Aplicación directa de conceptos propios de las Ciencias de la Computación:

- …
- Construcción de sistemas computacionales más elegantes y sencillos.
- Diseño (Maquina Secuencial --> Código)
- Diseño de estructuras y "parsing": gramaticas (ej: XML)
  - Búsqueda de cadenas o comparación de "patrones"
- SW para diseñar y evaluar circuitos digitales.
- "Escanear" grandes cantidades de texto (web)
- SW para verificar sistemas que tiene un número finito de "estados"

8

El por qué de la Teoría de Autómatas

- Teoría de la Computación:
  - ¿Aburrida y arcaica? NO, es Comprensible e Interesante.
- Proporciona al Ingeniero:
  - Aspectos teóricos (permite innovación)
    - Autómatas,
    - Representación Estructural (Gramáticas)
    - Autómatas y Máquinas para establecer los límites de la Computabilidad.
  - Aspectos prácticos (ingeniería)

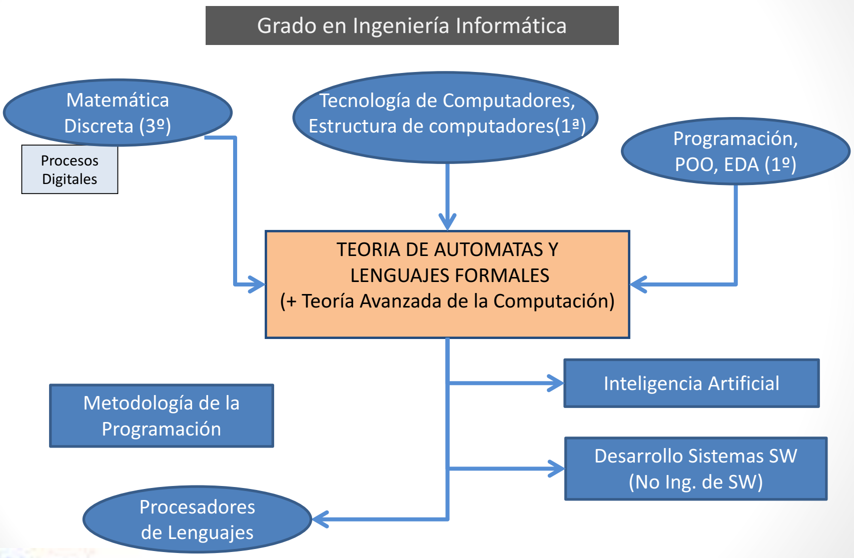
9

Índice

- El por qué de la Teoría de Autómatas
- Relación con otras Áreas de Conocimiento
- Máquinas, Lenguajes y Algoritmos

10

Relación con otras áreas.



11

Índice

- El por qué de la Teoría de Autómatas
- Relación con otras Áreas de Conocimiento
- Máquinas, Lenguajes y Algoritmos

12

Máquinas, Lenguajes y Algoritmos

Tres pilares sustentan la Teoría de Lenguajes, Gramáticas y Autómatas

AUTÓMATAS (ingeniería)
- Leonardo Torres, 1915
- Shannon, 1938
- Mc Culloch-Pitts, 1943
- Moore, 1956

COMPUTABILIDAD (matemáticas)
- Hilbert, 1928
- Gödel, Kleene, Post y Turing, ≈1930
- Church, 1936
- Rabin, 1960
- Cobhan, 1964
- Cook, 1972
- Aho, Hopcroft, Ullman, 1974

LENGUAJES y GRAMÁTICAS (lingüística)
- Panini, entre el 400 y 200 AC
- Chomsky, 1967
- Backus, ≈1960
- Kleene, 1951
- Hirst, Tennant y Carbonell, 1981

13

Máquinas, Lenguajes y Algoritmos

Máquinas o Autómatas

- Aplicación en campos muy diversos
- Manejan conceptos como “**control**”, “**acción**”, “**memoria**”
- Los objetos son controlados o recordados con símbolos, palabras o frases de algún tipo.

- Máquina de Moore y máquina de Mealy
- Circuitos combinatorios
- Autómatas Probabilísticos (incertidumbre en las transiciones)
- McCulloch-Pitts (1943) describieron los cálculos lógicos inmersos en un dispositivo denominado neurona artificial.
  - Redes de Neuronas Artificiales
- Autómatas Celulares (J.H. Conway, el juego de la vida).

14

Máquina de Turing Universal, Jim Wiked.

http://blogs.20minutos.es/mati-una-profesora-muy-particular/2012/09/10/1957/

15

Máquinas, Lenguajes y Algoritmos

Lenguajes y Gramáticas

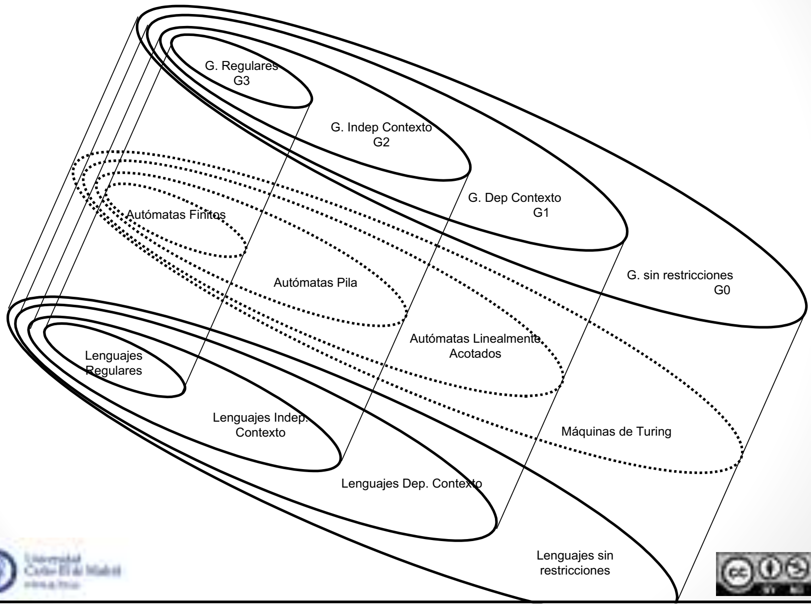
- Origen en la lingüística
- Noam Chomsky
  - Jerarquía de Chomsky (1956)
- "Backus normal form"
  (para gramática de ALGOL)
  - Lenguajes de Programación
  - Lenguajes Naturales
  - Sistemas de Comandos

Noam Chomsky
(1928 -)

John Backus
(1924 - 2007)

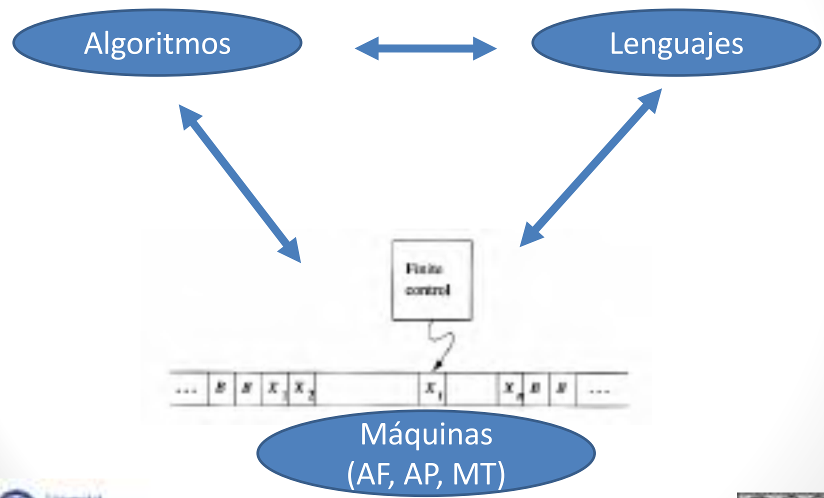
16

Máquinas, Lenguajes y Algoritmos



17

Máquinas, Lenguajes y Algoritmos



18

Bibliografía

- **Referencias básicas :**
  1. J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introducción a la Teoría de Autómatas, Lenguajes y Computación*. Ed. Pearson Addison Wesley , 2008
     Capítulo 1. Introducción a lo Autómatas
  2. E. Alfonseca Cubero, M. Alfonseca Moreno, R. Moriyón Salomón. Teoría de Autómatas y Lenguajes Formales. Ed. McGraw-Hill, 2007
     Capítulo 1. Máquinas, Lenguajes y Problemas.
- **Referencias complementarias:**
  1. P. Isasi, P. Martínez, D. Borrajo. Lenguajes, Gramáticas y Autómatas: Un enfoque práctico. Ed. Addison-Wesley, 1997
     Capítulo 2. Lenguajes y Gramáticas Formales
  2. D. M Kelley. Teoría de autómatas y lenguajes formales. Prentice-Hall, 1995
     Capítulo 2. Lenguajes Regulares.
  3. R. Penrose. La Nueva Mente del Emperador. DeBolsillo, 2011
     Capítulo 1. ¿Puede tener mente un computador?
     Capítulo 2. Algoritmos y máquinas de Turing
  4. R. Penrose. Las sombras de la mente: hacia una comprensión científica de la consciencia. Mondadori. 1996
  5. D.R. Hofstadter. Gödel, Escher, Bach : un eterno y grácil bucle. Tusquets, 1998

19

1. Introducción a la Teoría de Autómatas y Lenguajes Formales

Grado Ingeniería Informática
Teoría de Autómatas y Lenguajes Formales

20

Parte II

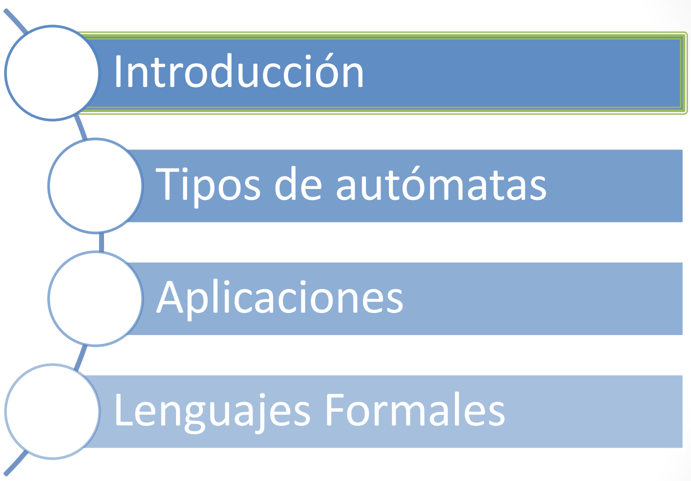
TEMA 2. Autómatas Formales

2.Teoría de Autómatas

Grado Ingeniería Informática
Teoría de Autómatas y Lenguajes Formales

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m



A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

2

uc3m

Introducción y definiciones

- Se trata de saber qué (y qué no) se puede computar.

  Y además…
  cómo de rápido,
  con cuánta memoria y
  con qué modelo de computación.

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

Introducción y definiciones

- Qué se entiende por **computación**?
- La Teoría de Autómatas se centra en la computación en sí, no en detalles sobre dispositivos de entrada y salida.
  (Así, no se trata de crear modelos matemáticos para un video juego, por ejemplo).

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

Autómata

Definición RAE

autómata.

(Del lat. automăta, t. f. de -tus, y este del gr. αὐτόματος, espontáneo).

1. **m. Instrumento o aparato que encierra dentro de sí el mecanismo que le imprime determinados movimientos**.
2. m. Máquina que imita la figura y los movimientos de un ser animado.
3. m. coloq. Persona estúpida o excesivamente débil, que se deja dirigir por otra.

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

Modelo Matemático

Autómata:

Modelo Matemático de computación.

Dispositivo abstracto con capacidad de computación.

Teoría de Autómatas:

Abstracción de cualquier tipo de computador y/o lenguaje de programación.

Desglose en sus elementos básicos (Entrada, Estado, Transición, Salidas y elementos auxiliares)

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

- Introducción
- Tipos de autómatas
- Aplicaciones
- Lenguajes Formales

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

Tipos de autómatas

- Autómatas Finitos (y máquinas secuenciales)
- Autómatas Probabilísticos
- Autómatas a Pila
- Células de Mc Culloch-Pitts
- Máquinas de Turing
- Autómatas Celulares
- Redes de Neuronas Artificiales

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

Tipos de autómatas

- Autómatas Finitos (
- Autómatas Probabi
- Autómatas a Pila
- Células de Mc Culloch-P
- Máquinas de Turing
- Autómatas Celulares
- Redes de Neuronas Artificiale

Turing estudió una máquina abstracta con la misma capacidad que los computadores actuales desde el punto de vista de lo que son capaces de hacer.

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

9

uc3m

Tipos de autómatas

- Autómatas Finitos (y máquinas secuenciales)
- Autómatas Probabilísticos
- Autómatas a Pila
- Células de Mc Culloch-Pitts
- Máquinas de Turing
- Autómatas Celulares
- Redes de Neuronas Artificiales

Mayor capacidad de cómputo.

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

10

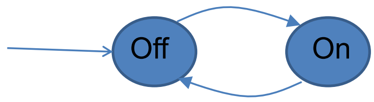
uc3m

Autómatas y Algoritmos

- La máquina de Turing es un modelo matemático abstracto que formaliza el concepto de algoritmo

- Todo Autómata puede ser transformado en un algoritmo y a la inversa.

Autómata Finito:



A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

Autómatas Discretos, continuos e híbridos

Criterio: Entradas

- Suelen ser DISCRETOS:
  - Autómatas Finitos (y máquinas secuenciales)
  - Autómatas a Pila
  - Máquinas de Turing

- Son DISCRETOS, CONTÍNUOS Y/O HÍBRIDOS:
  - Autómatas Celulares
  - Redes de Neuronas Artificiales

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

- Introducción
- Tipos de autómatas
- Aplicaciones
- Lenguajes Formales

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

13

uc3m

Aplicaciones de los autómatas

El Juego de la Vida

- Ejemplo de un juego implementado usando un **autómata celular**. Diseñado por el matemático británico John Conway en 1970.
- El todo es más que la suma de las partes.
- Las transiciones dependen del número de células vecinas vivas:
  - Una célula muerta con exactamente 3 células vecinas vivas "nace" (al turno siguiente estará viva).
  - Una célula viva con 2 ó 3 células vecinas vivas sigue viva, en otro caso muere o permanece muerta (por "soledad" o "superpoblación").

http://www.youtube.com/watch?v=XcuBvj0pw-E

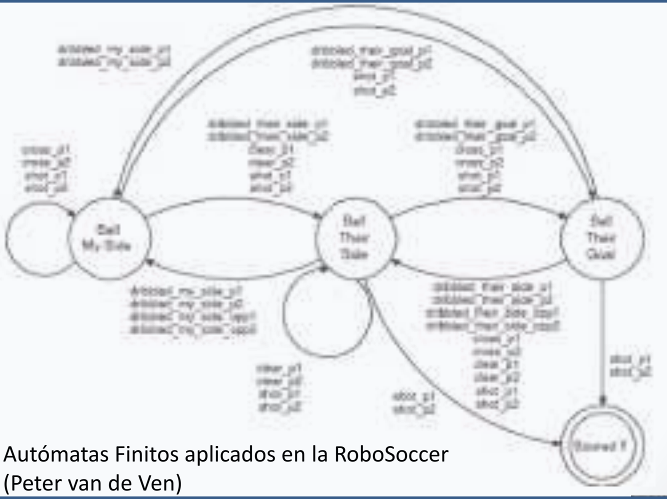
A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

14

uc3m

Aplicaciones de los autómatas

Comportamiento de robots en la RoboSoccer



Autómatas Finitos aplicados en la RoboSoccer
(Peter van de Ven)

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

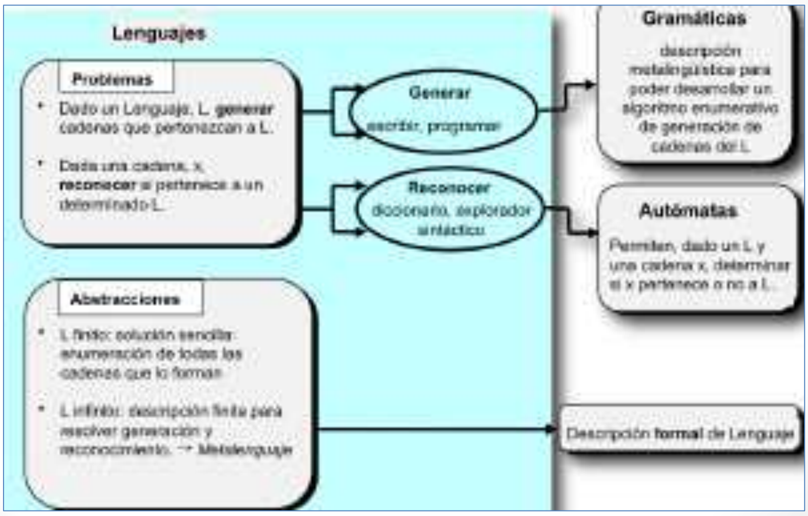
uc3m

- Introducción
- Tipos de autómatas
- Aplicaciones
- Lenguajes Formales

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

Lenguajes Formales



A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

Lenguajes Formales. Definiciones

Símbolo: Entidad abstracta, realmente no se define (análogo al punto en geometría). Son letras, dígitos, caracteres, etc. Forman parte de un alfabeto. También posible encontrar símbolos formados por varios caracteres, pej: IF, THEN, ELSE, ...

Alfabeto (Σ): Conjunto finito no vacío de letras o símbolos.

Sea "a" una letra y Σ un alfabeto, si a pertenece a ese alfabeto $\Rightarrow$ $a \in \Sigma$

Ejemplos:

- Σ1= {A, B, C, ...,Z} alfabeto de las letras mayúsculas
- Σ2= {0, 1} alfabeto binario
- Σ3= {IF, THEN, ELSE, BEGIN, END} alfabeto de símbolos para programación.

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

Lenguajes Formales. Definiciones

Palabra: toda secuencia finita de símbolos del alfabeto.

Σ_1= {A, B, C, ...,Z}; palabras sobre Σ_1 JUAN, ISABEL, etc.

Σ_2= {0, 1}; palabras sobre Σ_2 00011101

Σ_3= {IF, THEN, ELSE, BEGIN, END};

palabras sobre Σ_3 IFTHENELSEEND

Notación: se representan por letras minúsculas del final del alfabeto (*x, y, z*)

Ejem: *x*= JUAN; *y*= IFTHENELSEEND; z=00001111111111

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

Lenguajes Formales. Definiciones

Longitud de palabra: número de símbolos que componen una palabra.

Se representa por $|x|$

Ejemplos:

Σ_1= {A, B, C, ...,Z}; $|x|$ = |JUAN| = 4

$|y|$ = |IFTHENELSEEND| =13

Σ_3= {IF, THEN, ELSE, BEGIN, END};

$|y|$ = |IFTHENELSEEND| = 4 OJO!!!!

Palabra vacía λ: Es aquella palabra cuya longitud es cero

Se representa por λ, $|\lambda| = 0$

Sobre cualquier alfabeto es posible construir λ

Utilidad: será el elemento neutro en muchas operaciones (concatenación, etc.) con palabras y lenguajes

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

Lenguajes Formales. Definiciones

Universo del discurso, W(Σ): conjunto de todas las palabras que se pueden formar con los símbolos de un alfabeto Σ

También se denomina **Lenguaje Universal del alfabeto Σ**

Se representa como W(Σ)

Es un conjunto infinito (i.e. número infinito de palabras)

Ejemplo: sea Σ_4 = {A,B}, W(Σ_4) = {**λ,** A,B, AA,AB,BA,BB, AAA, ...} con un número ∝ de palabras

COROLARIO:

∀ Σ, λ ∈ W(Σ) ⇒ La palabra vacía pertenece a todos los lenguajes universales de todos los alfabetos posibles

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

Lenguajes Formales. Operaciones

Algunas operaciones importantes con palabras , sobre palabras de un universo del discurso dado:

Concatenación de palabras:

sean dos palabras *x*, *y* tal que $x \in W(\Sigma)$, $y \in W(\Sigma)$, y sea

$|x| = i = |x_1x_2...x_i|$ e $|y| = j = |y_1y_2...y_j|$,

se llama concatenación de x con y, a:

$x \cdot y = x_1x_2...x_i\, y_1y_2...y_j =$ **z**, donde $z \in W(\Sigma)$

Propiedades de la concatenación:

- Operación cerrada
- Propiedad Asociativa
- Con elemento neutro
- No conmutativa

Definiciones:

- cabeza
- cola
- longitud de palabra

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

Lenguajes Formales. Operaciones

Potencia de una palabra: Reducción de la concatenación a los casos que se refieren a una misma palabra

- potencia *i-ésima* de una palabra es el resultado de concatenar esa palabra consigo misma *i* veces
- La concatenación es asociativa $\Rightarrow$ no especificar el orden
- $x^i = x \cdot x \cdot x \cdot \ldots \cdot x$ (“x” i veces)
- $|x^i| = i \cdot |x|$ (i>0)
- se cumple:
  - $x^1 = x$
  - $x^{1+i} = x \cdot x^i = x^i \cdot x$ (i>0)
  - $x^{j+i} = x^j \cdot x^i = x^i \cdot x^j$ (i, j>0)
- Si se define $x^0 = \lambda$

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

23

uc3m

Lenguajes Formales. Definiciones

Lenguaje (L): Se denomina lenguaje sobre el alfabeto Σ:

- a todo subconjunto del lenguaje universal de Σ, $L \subset W(\Sigma)$
- a todo conjunto de palabras sobre un determinado Σ (generado a partir del alfabeto Σ)

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

24

uc3m

Lenguajes Formales

Lenguajes Especiales:

1. **ϕ** = Lenguaje vacío, $\phi \subset W(\Sigma)$
2. **$\{\lambda\}$** = Lenguaje de la palabra vacía
   - se diferencian en el número de palabras (cardinalidad) que los forman $C(\phi) = 0$ mientras que $C(\{\lambda\})=1$
   - se parecen en que ϕ y $\{\lambda\}$ son lenguajes sobre cualquier alfabeto
3. Un alfabeto es uno de los lenguajes generados por el mismo:

   $\Sigma \subset W(\Sigma)$, por ejemplo el chino

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

Lenguajes Formales

Concatenación de Lenguajes : Sobre un alfabeto dado Σ

Sean L_1 y L_2 definidos sobre el mismo alfabeto Σ, L_1, $L_2 \subset W(\Sigma)$;

se llama **concatenación** de dos lenguajes, L_1, L_2 y se representa por $L_1 . L_2$ al lenguaje así definido:

$$L_1 . L_2 = \{ x.y \ / \ x \in L_1 \text{ AND } y \in L_2 \}$$

Es el conjunto formado por palabrasformadas por parte del del primero seguidas de parte del segundo.

Operación cerrada   Asociativa Elemento Neutro $\{\lambda\}$ NO comuntativa

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

Lenguajes Formales

Unión de Lenguajes : Sobre un alfabeto dado Σ

Sean L_1 y L_2 definidos sobre el mismo alfabeto Σ, L_1, $L_2 \subset W(\Sigma)$;

se llama **unión** de dos lenguajes, L_1, L_2 y se representa por $L_1 \cup L_2$ al lenguaje así definido:

$$\mathbf{L_1 \cup L_2 = \{ x \,/\, x \in L_1 \text{ ó } x \in L_2 \} =}$$

Es el conjunto formado indistintamente por palabras de uno u otro de los dos lenguajes (equivale a la suma). $\mathbf{L_1 + L_2 = L_1 \cup L_2}$

Op. Cerrada, Asociativa, Conmutativa y Elemento neutro ϕ

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

2.Teoría de Autómatas

Araceli Sanchis de Miguel
Agapito Ledezma Espino
José A. Iglesias Martínez
Beatriz García Jiménez
Juan Manuel Alonso Weber

Grado Ingeniería Informática
Teoría de Autómatas y Lenguajes Formales

A. Sanchis, A. Ledezma, J.A. Iglesias, B. García, J. M.Alonso

uc3m

Parte III

TEMA 3. Autómatas Finitos

3. Autómatas Finitos

Grado Ingeniería Informática
Teoría de Autómatas y Lenguajes Formales

Objetivos

- Definir el concepto de Autómata Finito Determinista (AFD).
- Definir el concepto de Autómata Finito No Determinista (AFND).
- Establecer las equivalencias entre AFD.
- Convertir un AFND en un AFD.
- Minimizar AFD.
- Identificar el tipo de lenguaje aceptado por un AFND.

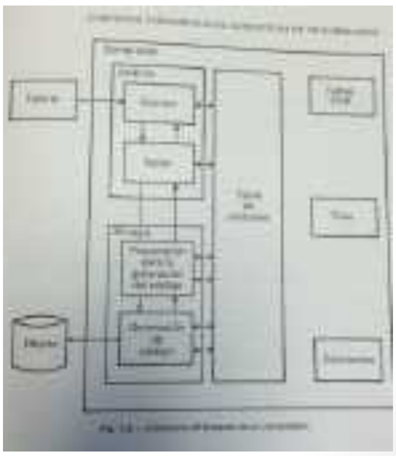
2

- Autómatas Finitos
  - Autómatas Finitos Deterministas (AFD)
    - AFD como reconocedores de lenguajes
    - Equivalencia y minimización de AFD
  - Autómatas Finitos No Deterministas (AFND)
- Equivalencia entre AFD y AFND

3

Aplicaciones de AFs

- **Compilador:** Proceso de traducción que convierte un programa fuente escrito en un lenguaje de alto nivel a un programa objeto en código máquina y listo para ejecutarse en en ordenador –con poca o ninguna preparación adicional.
- **Fases**: análisis y síntesis



4

Autómatas Finitos

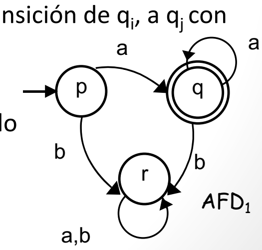
- Los Autómatas Finitos son de dos tipos:
  - **Deterministas:**
    - cada combinación (estado, símbolo de entrada) produce un solo (estado).
  - **No Deterministas:**
    - cada combinación (estado, símbolo de entrada) produce varios (estado1, estado 2, ..., estado i).
    - son posibles transiciones con λ

5

Autómatas Finitos. Representación

- Se pueden representar mediante:
  1. Diagramas de transición o
  2. Tablas de transición

1. **Diagramas de transición:**
   - Nodos etiquetados por los estados ($q_i \in$ Conjunto de estados)
   - Arcos entre nodos q_i a q_j etiquetados con e_i (e_i es un símbolo de entrada) si existe la transición de q_i, a q_j con e_i
   - El estado inicial se señala con $\rightarrow$
   - El estado final se señala con * o doble círculo



6

Autómatas Finitos. Representación

2. Tablas de transición:

- Filas encabezadas por los estados ($q_i \in$ Conjunto de estados)
- Columnas encabezadas por los símbolos de entrada ($e_i \in$ alfabeto de entrada)

Símbolos de Entrada

Estados

	e_1	e_2	…	e_n
→ q_1		$f(q_1, e_2)$		
…				
$*q_m$				

7

- Autómatas Finitos
- Autómatas Finitos Deterministas (AFD)
- AFD como reconocedores de lenguajes
- Equivalencia y minimización de AFD
- Autómatas Finitos No Deterministas (AFND)
- Equivalencia entre AFD y AFND

8

Autómatas Finitos Deterministas

- AF Deterministas, **AFD**'s: se definen mediante una quíntupla

  (Σ, Q, f, q_0, F), donde:

  - **Σ: alfabeto de entrada**
  - **Q: conjunto de estados**, es conjunto finito no vacío, realmente un alfabeto para distinguir a los estados
  - **f: Q$\times\Sigma \rightarrow$ Q, función de transición**
  - **$q_0 \in$Q, estado inicial**
  - **F$\subset$Q: conjunto de estados finales o de aceptación**

9

Autómatas Finitos Deterministas

- Ejemplo: El AFD_1 = ({0,1}, {p,q,r}, f, p, {q}), donde f está definida por:

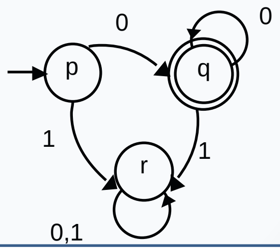
  f(p,0) = q   f(p,1) = r

  f(q,0) = q   f(q,1) = r

  f(r,0) = r   f(r,1) = r

Tiene la tabla de transición y el diagrama de estados siguientes:

	0	1
→ p	q	r
*q	q	r
r	r	r



10

- Autómatas Finitos
- Autómatas Finitos Deterministas (AFD)
- **AFD como reconocedores de lenguajes**
- Equivalencia y minimización de AFD
- Autómatas Finitos No Deterministas (AFND)
- Equivalencia entre AFD y AFND

AFD como reconocedores de Lenguajes

- Cuando un AF transita desde q_0 a un estado final en varios movimientos, se ha producido el RECONOCIMIENTO o ACEPTACIÓN de la cadena de entrada

- Cuando un AF no es capaz de alcanzar un estado final, se dice que el AF NO RECONOCE la cadena de entrada y que ésta NO PERTENECE al lenguaje reconocido por el AF

AFD. Conceptos Básicos

- **Configuración**: es un par ordenado de la forma (q,w) donde:
  - q: estado actual del AF
  - w: cadena que le queda por leer en ese instante, w $\in \Sigma^*$ Universo del Discurso
- Configuración inicial: (q_0, t)
  - q_0: estado inicial
  - t: cadena de entrada a reconocer por el AFD, t $\in \Sigma^*$
- Configuración final: (q_i,λ)
  - q_i: estado final
  - λ la cadena de entrada ha sido leída completamente

- **Movimiento:** es el tránsito entre dos configuraciones.

$$(q,aw) \xrightarrow[f(q,a)=q']{} (q',w) \qquad a,w \in \Sigma^*$$

13

AFD. Conceptos Básicos

Extensión a palabra de la función de transición f, f':

Es la ampliación de la definición de **f** a palabras de Σ*, i.e. w ∈ Σ*

- f': Q x Σ* → Q

  a partir de f, que sólo considera palabras de longitud 1,

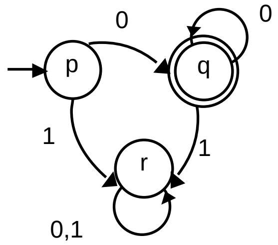
  hay que añadir:

  - $f'(q,\lambda) = q \quad \forall q \in Q$
  - $f'(q, a\cdot x) = f'(f(q,a), x) \quad \forall q \in Q,$ $\boxed{a \in \Sigma, x \in \Sigma^*}$

14

AFD. Conceptos Básicos

- En el AFD_1 (de la figura), indicar el resultado de las siguientes expresiones:
  - f'(p,λ)
  - f'(p, 0^n)
  - f'(p,11)
  - f'(p,0011010)
  - f' (p,100)



15

AFD. Conceptos Básicos

Lenguaje asociado a un AFD:

- Sea un AFD = (Σ, Q, f, q_0, F), se dice que una palabra x es aceptada o **reconocida** por el AFD si f' (q_0,x) ∈ F
- Se llama **lenguaje asociado a un AFD** al conjunto de todas las palabras aceptadas por éste:

  L = { x / x ∈ Σ * and f' (q_0,x) ∈ F }

  - Si F = { } = Ø ⇒ L= Ø
  - Si F = Q ⇒ L= Σ *

  - Otra definición:

    L = { x / x ∈ Σ * and (q_0, x) → (q,λ) and q ∈ F}

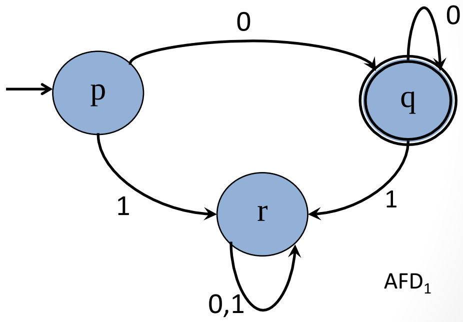
16

AFD. Conceptos Básicos

En el AFD_1

- Cuál es $L(AFD_1)$ = ¿?

- Y si se hace F = {r}, cuál es $L(AFD_1)$ = ¿?

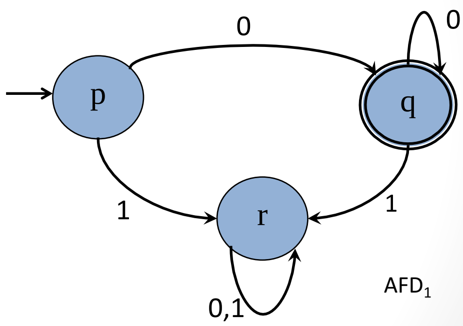


AFD_1

AFD. Conceptos Básicos

En el AFD_1

- Cuál es $L(AFD_1)$ = **$\{0^n / n > 0\}$.**

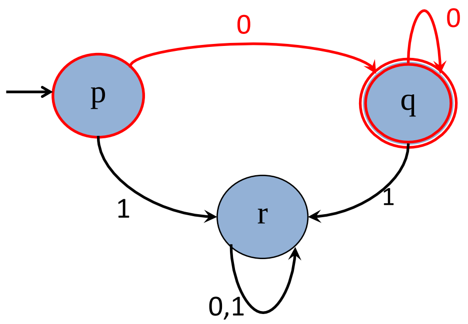


AFD_1

AFD. Conceptos Básicos

En el AFD_1

- Cuál es $L(AFD_1) = \{0^n \,/\, n > 0\}$. **Comprobación**

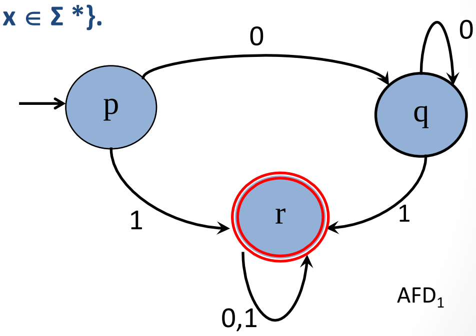


Desde p, con el número de 0's que sea, pero siempre al menos uno, se llega al estado final

AFD. Conceptos Básicos

En el AFD_1

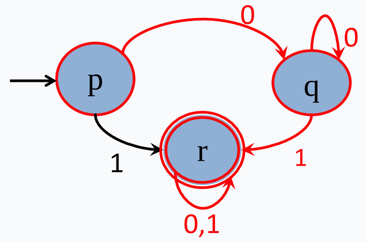
- Y si se hace $F = \{r\}$,
  $L(AFD_1) = \{0^n1x \,/\, n \geq 0,\, x \in \Sigma^*\}$.



AFD. Conceptos Básicos

En el AFD_1,

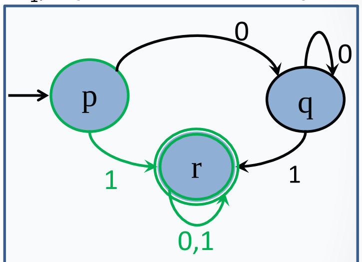
- comprobar que si se hace F = {r}, $L(AFD_1)$ = **{0^n1x / $n \geq 0, x \in \Sigma^*$}.**



Desde p, con un "0" llego al estado q y desde allí se pueden aceptar tantos 0s como sean.

Luego con un 1 salto al estado final y allí puedo terminar o reconocer cualquier cadena de 0s y 1s.

Expresión regular: L_A= 0^+1 (0+1)*



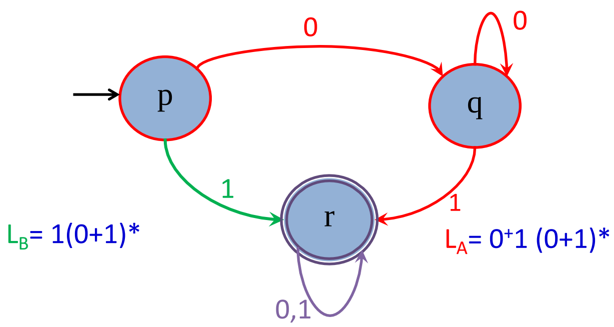
Expresión regular: L_B= 1(0+1)*

21

AFD. Conceptos Básicos

En el AFD_1,

- comprobar que si se hace F = {r}, $L(AFD_1)$ = **{0^n1x / $n \geq 0, x \in \Sigma^*$}.**



Expresión regular L_A U L_B = 0*1(0+1)*

22

AFD. Conceptos Básicos

Estados accesibles y Autómatas conexos:

- Sea un AFD = (Σ, Q, f, q_0, F), el estado p $\in$ Q es ACCESIBLE desde q $\in$ Q si $\exists$ x $\in$ Σ^* f'(q,x) = p. En otro caso se dice que INACCESIBLE.
  Todo estado es accesible desde sí mismo pues f'(p,λ) = p

Teoremas:

- teorema 3.2.2, libro 1 de la bibliografía.
  Sea un AFD, $|Q| = n$, $\forall$ p, q $\in$ Q p es accesible desde q
  sii $\exists$ x$\in\Sigma^*$, $|x| < n$ / **f'(q,x) = p**
- teorema 3.2.3, libro 1 de la bibliografía
  Sea un AFD, $|Q| = n$, entonces $L_{AFD} \neq \phi$ **sii** el AFD acepta al menos una palabra x$\in\Sigma^*$, $|x| < n$
  Nota: sii= "si y solo si"

23

AFD. Conceptos Básicos

Estados accesibles y **Autómatas conexos:**

Sea un AFD = (Σ, Q, f, q_0, F). Diremos que el autómata es conexo si todos los estados de Q son accesibles desde q_0

Dado un autómata no conexo, podemos obtener a partir de él otro autómata equivalente conexo eliminando los estados inaccesibles desde el estado inicial. Los autómatas reconocen el mismo lenguaje.

Eliminación de estados inaccesibles.

- *¿Qué algoritmo, para ser implementado en un programa, se podría implementar para marcar los accesibles?*

24

AFD. Ejercicios

- Hallar el AFD conexo equivalente al dado: AF= ({0,1}, {p,q,r,s}, p, f, {q,r,s}), donde f viene dada por la tabla.
  - Se eliminan todos los estados innacesibles y todos las transiciones (i.e. arcos) que salen desde dichos estados innacesibles.

	0	1
p	r	p
*q	r	p
*r	r	p
*s	s	s

- Indicar, además el leguaje reconocido por ambos AFD's (original y conexo).

- Autómatas Finitos
- Autómatas Finitos Deterministas (AFD)
- AFD como reconocedores de lenguajes
- Equivalencia y minimización de AFD
- Autómatas Finitos No Deterministas (AFND)
- Equivalencia entre AFD y AFND

AFD. Equivalencia y Minimización

- Es posible tener varios autómatas que reconozcan el mismo lenguaje.
- Para todo autómata se puede obtener un autómata equivalente (i.e. reconoce el mismo lenguaje) donde el número de estados del autómata sea el mínimo.
- **¿Por qué interesa obtener el mínimo?** **(Apartado 4.4 Libro 2 bibliografía)**

27

AFD. Equivalencia y Minimización

¿Por qué interesa obtener el AFD mínimo? (Ap. 4.3 y 4.4 Libro 2 bibliograf)

- Se dispone de un descriptor del lenguaje (lenguaje regular): gramática tipo 3, AFD, AFND, expresión regular.
- Se plantean problemas de decisión:
  - ¿El lenguaje descrito es vacio?
  - ¿Existe una determinada cadena w en el lenguaje descrito?
  - ¿Dos descripciones de un lenguaje describen realmente el mismo lenguaje?
    - Nota: usualmente los lenguajes son infinitos, con lo que no es posible plantear la pregunta y recorrer el conjunto INFINITO de cadenas.
- Los algoritmos para responder a las dos primeras preguntas son sencillos. ¿Pero y para la última pregunta ?

  ¿Dos descripciones de un lenguaje describen realmente el mismo lenguaje?

  Consecuencia de esta comprobación: es necesario obtener el AFD mínimo equivalente

28

AFD. Equivalencia y Minimización

Teoremas:

- Equivalencia de estados:

  p E q, donde $p,q \in Q$, si $\forall x \in \Sigma^*$ se verifica que

  $f'(p,x) \in F \Leftrightarrow f'(q,x) \in F$

- Equivalencia de orden (o de longitud) "n"

  p E_n q, donde $p,q \in Q$, si $\forall x \in \Sigma^* / |x| \leq n$ se verfica que

  $f'(p,x) \in F \Leftrightarrow f'(q,x) \in F$

E y E_n son relaciones de equivalencia.

AFD. Equivalencia y Minimización

Equivalencia de estados – Casos particulares:

- **E_0**, x palabra $|x| \leq 0 \Rightarrow x = \lambda$ se verifica que

  p E_0 q, $\forall p,q \in Q$, si $\forall x \in \Sigma^* / |x| \leq 0$ se verfica que

  $f'(p,x) \in F \Leftrightarrow f'(q,x) \in F$

  x es lamba

  $f'(p,x) = f'(p,\lambda) = p$ (por definición de f')

  $f(p, \lambda) \in F \Leftrightarrow f(q, \lambda) \in F$ -> $p \in F \Leftrightarrow q \in F$

  Todos los estados finales de son E_0 equivalentes.

  $\forall$ $p,q \in F$ se cumple que p E_0 q

  $\forall$ $p,q \in Q - F$ se cumple que p E_0 q

AFD. Equivalencia y Minimización

Equivalencia de estados – Casos particulares:

- E_1, x palabra $|x| \leq 1$, $(x \in \Sigma)$ se verifica que

  p E_1 q, $\forall$ p,q $\in$ Q, si $\forall$ x$\in$ Σ^* / $|x| \leq 1$ se verifica que

  $f'(p,x) \in F \Leftrightarrow f'(q,x) \in F$

  x es lamba o símbolo del alfabeto.

  $f'(p,x) = f'(p,a) = f(p,a)$ ó $f'(p,x) = f'(p,\lambda) = p$ (por definición de f')

  $f(p,a) \in F \Leftrightarrow f(q,a) \in F$

  Partiendo de p y q con una sola transición se debe llegar a un estado final para ambos casos o uno no final para ambos casos.

31

AFD. Equivalencia y Minimización

- **Propiedades** Nota: en estas expresiones matemáticas, "n" *no* significa |Q|
  - Lema: $p \, E \, q \Rightarrow p \, E_n \, q$, $\forall$ n, p, q$\in$ Q
  - Lema: $p \, E_n \, q \Rightarrow p \, E_k \, q$, $\forall$ n > k
  - Lema: $p \, E_{n+1} \, q \Leftrightarrow p \, E_n \, q$ and $f(p,a) \, E_n \, f(q,a)$ $\forall$ a $\in \Sigma$

- **Teorema**: $p \, E \, q \Leftrightarrow p \, E_{n-2} \, q$, donde $n = |Q| > 1$ Aquí "n" *sí* significa |Q|

  (Teorema 5.1 (pag 117 libro 4 bibliografía))

  p E q sii $\forall$ x$\in\Sigma^*$, $|x| = m \leq n-2$ se verifica que $f(p,x) \in F \Leftrightarrow f(q,x) \in F$

  m = n-2 es el valor más pequeño que cumple este teorema

  (n-1 sí lo cumple, pero n-3 no se garantiza que se cumpla)

32

AFD. Equivalencia y Minimización

"E" es una relación de equivalencia. ¿Qué significa Q/E?

- **Q/E es una partición de Q,**
- **$Q/E = \{C_1, C_2, \ldots, C_m\}$, donde $C_i \cap C_j = \emptyset$**
  - p E q $\Leftrightarrow$ ($p,q \in C_i$), por lo tanto

    $\forall x \in \Sigma^*$ se verifica que $f'(p,x) \in C_i \Leftrightarrow f'(q,x) \in C_i$

Nota: en libro 1 biblio, $p,q \in Ci$ se representa por $p = q = C_i$;

- Para la relación de orden n
  - E_n: $Q/E_n = \{C_1, C_2, \ldots, C_m\}$, C_i intersección $C_j = \emptyset$
  - $p\, E_n\, q \Leftrightarrow p,q \in C_i$;
  - por lo tanto $\forall x \in \Sigma^*$, $\underline{|x| \leq n}$ se verifica que $f'(p,x) \in C_i \Leftrightarrow f'(q,x) \in C_i$

33

AFD. Equivalencia y Minimización

Propiedades. (Lemas)

- Lema: Si $Q/E_n = Q/E_{n+1} \Rightarrow Q/E_n = Q/E_{n+i}$ $\forall$ i = 0, 1, ...
- Lema: Si $Q/E_n = Q/E_{n+1} \Rightarrow Q/E_n = Q/E$ conjunto cociente
- Lema: Si $|Q/E_0| = 1 \Rightarrow Q/E_0 = Q/E_1$
- Lema: $n = |Q| > 1 \Rightarrow Q/E_{n-2} = Q/E_{n-1}$
- $p\, E_{n+1}\, q \Leftrightarrow (p\, E_n\, q$ and $f(p,a)\, E_n\, f(q,a)\ \forall a \in \Sigma)$

34

AFD. Equivalencia y Minimización

Interpretación lemas anteriores:

El objetivo es obtener la partición Q/E, puesto que será el autómata mínimo, sin estados equivalentes .

- En cuanto se obtienen dos particiones consecutivas $Q/E_k = Q/_{Ek+1}$, se para.
- Para obtener Q/E, hay que empezar por Q/E_0, Q/E_1, etc.
- Para obtener Q/E, hay que obtener Q/E_{n-2} en el peor caso, ya que si se obtiene $Q/E_{n-k} = Q/E_{n-k+1}$, con k>=3, se habría obtenido ya Q/E.
- El lema $p\ E_{n+1}\ q \Leftrightarrow p\ E_n\ q$ and $f(p,a)\ E_n\ f(q,a)\ \forall\ a \in \Sigma$, permite es extender la equivalencia de orden n desde E_0 y E_1

35

AFD. Equivalencia y Minimización

❑ **Teorema**:

$$pEq \Leftrightarrow pE_{n-2}\,q \text{ donde } |Q| = n > 1 \ (**)$$

Es decir, p E q Sii $\forall\ x \in \Sigma^*,\ |x| \leq n-2\ ,\ f'(p,x) \in F \Leftrightarrow f'(q,x) \in F$

n-2 es el valor más pequeño que cumple este teorema

36

AFD. Equivalencia y Minimización

Algoritmo formal para obtener Q/E:

1. **Q/E_0 = { F, no F}**
   1ª división en función de si son o no estados finales.

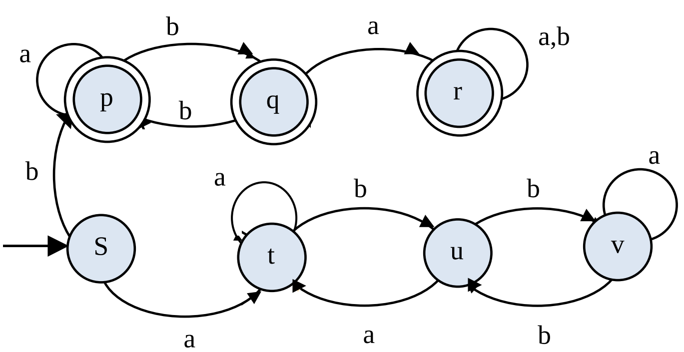
2. **Q/E_{i+1}**
   partiendo de **Q/E_i** = $\{C_1, C_2, \ldots C_n\}$, se construye **Q/E_{i+1}**:
   p y q están en la misma clase si:
   $p, q \in C_k \in Q/E_i \; \forall a \in \Sigma \Rightarrow f(p,a) \text{ y } f(q,a) \in C_m \in Q/E_i$

3. Si **$Q/E_i = Q/E_{i+1}$** entonces **$Q/E_i = Q/E$**
   Si no, repetir el paso 2 partiendo de Q/E_{i+1}

37

AFD. Equivalencia y Minimización

- Ejercicio: Hallar el AFD mínimo equivalente



38

AFD. Equivalencia

Autómatas Equivalentes:

- **Estados equivalentes en AFD's distintos**:
  - Sean 2 AFD's: (Σ,Q,f,q_0,F) y $(\Sigma',Q',f'' q_0',F')$
  - Los estados p,q / $p\in Q$ y $q\in Q'$ son equivalentes (pEq) si se verifica que $f''(p,x) \in F \Leftrightarrow f''(q,x) \in F' \;\; \forall x \in \Sigma^*$

- **Estados equivalentes en AFD's distintos:**
  - Dos AFD's son equivalentes si reconocen el mismo lenguaje, es decir: Si $f(q_0,x) \in F \Leftrightarrow f(q_0',x) \in F' \;\; \forall x \in \Sigma^*$. Es decir:
  - **Dos AFD's son equivalentes si lo son sus estados iniciales: q_0 E q_0'**

39

AFD. Equivalencia

¿Qué es la suma directa de 2 AFD's?

Sean 2 AFD's:

$$\left.\begin{array}{l} A1 = (\Sigma,Q_1,f_1,q_{01},F_1) \\ A2 = (\Sigma',Q_2,f_2,q_{02},F_2) \end{array}\right\} \boxed{\textbf{Donde } Q_1 \cap Q_2 = \phi}$$

y $\Sigma = \Sigma'$

Se llama **suma directa de A1 y A2** al AF A:

$A = A1 + A2 = (\Sigma, Q_1 \cup Q_2, f, q_0, F_1 \cup F_2)$, donde:

q_0 es el estado inicial de uno de los AF's

f: $f(p,a) = f1(p,a)$ si $p \in Q_1$

$f(p,a) = f2(p,a)$ si $p \in Q_2$

40

AFD. Equivalencia

❑ **Teorema:** (el teorema (**) aplicado a la suma directa de dos autómatas):

sean A1, A2 / $Q_1 \cap Q_2 = \phi$, $|Q_1| = n_1$, $|Q_2| = n_2$

A_1 E A_2 si q_{01} E q_{02} en A = A_1+A_2

Es decir, si A_1 y A_2 aceptan las mismas palabras x / $|x| \leq n_1+n_2-2$

además, n_1+n_2-2 es el valor mínimo que cumple el teorema

41

AFD. Equivalencia

Autómatas equivalentes, comprobación:

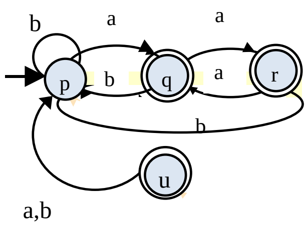
Algoritmo para comprobar la equivalencia de AFDs

1. Se hace la suma directa de los dos AFD's
2. Se hace Q/E del AFD suma
3. Si los dos estados iniciales están en la misma clase de equivalencia de Q/E $\Rightarrow$ los 2 AFD's son equivalentes

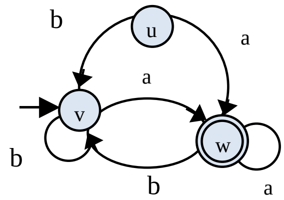
42

AFD. Equivalencia

- **Ejercicio:** Comprobar que los autómatas A1 y A2 son equivalentes.



A1

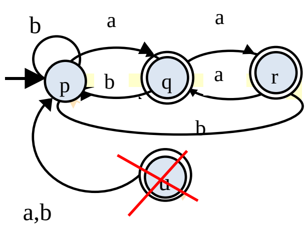


A2

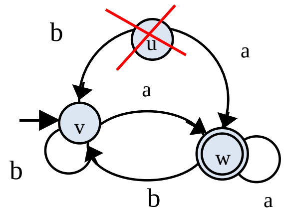
43

AFD. Equivalencia

- **Ejercicio:** Comprobar que los autómatas A1 y A2 son equivalentes.



A1



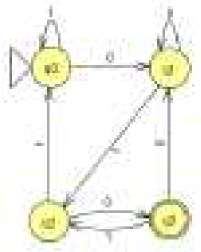
A2

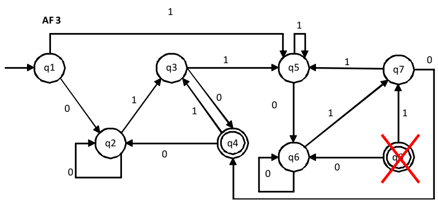
Los estados u de ambos autómatas son inaccesibles y habría que eliminarlos antes de ver si son equivalentes.

44

AFD. Equivalencia

- Ejercicio 7 de la hoja 2 de ejercicios: Comprobar si los dos AFD son equivalentes (obteniendo el mínimo para cada uno).





45

AFD. Equivalencia

- Ejercicio 7 de la hoja 2 de ejercicios: AF1 es mínimo. AF2:
- Q/E0 = {{q4,~~q8~~}, {q1,q2,q3,q5,q6,q7}} = C1, C2 **OJO, Q8 es inaccesible y habría que quitarlo**. Aparece tachado en la solución.
- Q/E1={{q4,~~q8~~}, {q1,q2,q5,q6}, {q3,q7}}= C1,C2,C3
- Q/E2 = {{q4,~~q8~~}, {q1,q5}, {q2,q6}, {q3,q7}}= C1,C2,C3,C4
- Q/E3 = Q/E2 = {{{q4,~~q8~~}, {q1,q5}, {q2,q6}, {q3,q7}}= C1,C2,C3,C4

	0	1	0	1	0	1	0	1
->q1	q2	q5	C2	C2	C2	C2	C3	C2
q2	q2	q3	C2	C2	C2	C3	C3	C4
q3	q4	q5	C1	C2	c1	C2	c1	C2
*q4	q2	q3	C2	C2	C2	C3	C3	C4
q5	q6	q5	C2	C2	C2	C2	C3	C2
q6	q6	q7	C2	C2	C2	C3	C3	C4
q7	q4	q5	c1	C2	c1	C2	c1	C2
~~*q8~~	~~q6~~	~~q7~~	~~C2~~	~~C2~~	~~C2~~	~~C3~~	~~C3~~	~~C4~~

- El AF2 y el AF1 son ISOMORFOS y por tanto son equivalentes.

46

AFD. Equivalencia

- Sean dos autómatas:
  - $A1 = (\Sigma, Q_1, f_1, q_{01}, F_1)$ y $A2 = (\Sigma', Q_2, f_2, q_{02}, F_2)$, tales que $|Q_1| = |Q_2|$
- Se dice que A1 y A2 **son isomorfos**, si existe una aplicación biyectiva $i : Q_1 \rightarrow Q_2$ que cumple:
  1. $i(q_{01}) = q_{02}$, es decir, los estados iniciales son correspondientes
  2. $q \in F_1 \Leftrightarrow i(q) \in F_2$ es decir, los estados finales son correspondientes
  3. $i(f_1(q,a)) = f_2(i(q),a)\ \forall\ a \in \Sigma\ \ q \in Q_1$

  - En definitiva, a cada estado le corresponde otro equivalente que solo se diferencia en el nombre de sus estados.
- Dos AFDs isomorfos, también son equivalentes y reconocen el mismo **lenguaje.**

47

AFD. Minimización

Sea el AFD, $A = (\Sigma, Q, f, q_0, F)$:

1. Partir del AFD conexo, i.e. eliminar estados inaccesibles desde el estado inicial
2. Construir Q/E del autómata conexo
3. El AFD mínimo, salvo isomorfismos, es:

   $A' = (\Sigma, Q', f', q_0', F')$

donde:

$Q' = Q/E$

f' se construye: $f'(C_i,a) = C_j$ si $\exists q \in C_i$, $p \in C_j$ / $f(q,a) = p$

$q_0' = C_0$ si $q0 \in C_0$, $C_0 \in Q/E$

$F' = \{C$ / C contiene al menos un estado de F($\exists$ un $q \in F$ tal que $q \in C)\}$

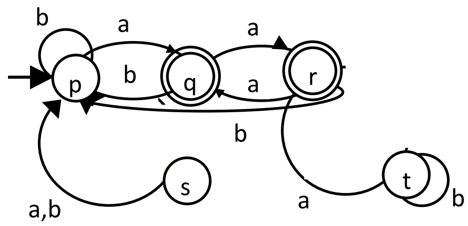
COROLARIO:

2 AFD's son equivalentes si sus AF mínimos respectivos son isomorfos.

48

AFD. Ejercicio

Hallar el AFD mínimo equivalente al dado:



- Autómatas Finitos
- Autómatas Finitos Deterministas (AFD)
- AFD como reconocedores de lenguajes
- Equivalencia y minimización de AFD
- Autómatas Finitos No Deterministas (AFND)
- Equivalencia entre AFD y AFND

Autómatas Finitos No Deterministas

Definiciones de AFND :

1. AFND = (Σ, Q, f, q_0, F), donde
   - f: Q x (Σ U λ }→ Q es No determinista,

   es decir, por ejemplo: f(p,a) = {q,r} y f(p,λ) = {q,r}

2. AFND = $(\Sigma, Q, f, q_0, F, T)$, donde
   - f : Q x Σ → *P*(Q): conjunto de las partes de Q
   - T : Relación definida sobre pares de elementos de Q.

   pTq = (p,q) ∈T si está definida la transición f(p, λ)=q

   Nota: "T" es la definición formal de la transición λ

51

Autómatas Finitos No Deterministas

Ejemplo: Sea el AFND siguiente:

A = ({a,b}, {p,q,r,s}, f,p, {p,s}, T= {(q,s), (r,r), (r,s), (s,r)}) donde f:

f(p,a) = {q}	f(p,b) = {}
f(q,a) = {p,r,s}	f(q,b) = {p,r}
f(r,a) = {}	f(r,b) = {p,s}
f(s,a) = {}	f(s,b) = {}

La tabla de transiciones es

	a	b	λ
→*p	q		
q	{p,r,s}	p,r	s
r		p,s	r,s
*s			r

52

AFNDs. Función de Transición extendida a palabras

- Se define a partir de f, una función de transición f”, que actúa sobre palabras de Σ^*;

  f” es la función de transición sobre palabras.

- Es una aplicación: f”: $Q \times \Sigma^* \rightarrow P(Q)$. Donde :

1. $f''(q,\lambda) = \{p \,/\, qT^*p \ \forall q \in Q\}$ (T* se define más adelante)

   donde se cumple que $q \in f'(q,\lambda)$

2. sea $x = a_1a_2a_3 \ldots a_n$, n>0

   $f''(q,x) = \{p$ / p es accesible desde q por medio de la palabra $\underbrace{\lambda^*a_1\,\lambda^*a_2\,\lambda^*a_3\,\lambda^* \ldots\, \lambda^*a_n\,\lambda^*}_{\text{es idéntica a x}}\ \forall q \in Q\}$

<u>**Lectura recomendada: Apartado 3.3.4 del primer libro de la bibliografía básica**</u>

53

AFNDs. Función de Transición extendida a palabras

Calculo de T*

Sea AFND = $(\Sigma, Q, f, q_0, F, T)$.

- Para calcular f” es necesario extender las transiciones con una λ a λ^*, es decir calcular T* del AFND= $(\Sigma, Q, f, q_0, F, \underline{\boldsymbol{T}})$
- Para ello existe el método formal de las matrices booleanas, o el método de la matriz de pares (estado, estado).

54

AFNDs. Función de Transición extendida a palabras

Calculo de T*. Método de la matriz de pares de estados

- Se construye una matriz con tantas filas como estados.
- En la 1ª columna se coloca el par correspondiente al estado en cuestión, es decir, por ej. (p,p) puesto que cada estado es accesible desde si mismo.
- En las columnas siguientes se añaden las transiciones λ definidas en el AFND, considerando si el hecho de añadirlas permite extender alguna transición más.
  - Pej. Si existe la transición λ (q,r) y se añade la transición λ (r,s), habrá que añadir asimismo, la transición (q,s).
- Cuando no sea posible añadir ningún par más, se habrá terminado T*

55

AFNDs. Función de Transición extendida a palabras

Calculo de T*. Ejemplo 1:

- Sea el AFND: A, definido anteriormente donde T= {(q,s), (r,r), (r,s), (s,r)}. Se trata de calcular T*

	a	b	λ
→ *p	q		
q	{p,r,s}	p,r	s
r		p,s	r,s
*s			r

$$T^* = \begin{pmatrix} (p,p) & & \\ (q,q) & (q,s) & \boxed{(q,r)} \\ (r,r) & (r,s) & \\ \boxed{(s,s)} & (s,r) & \end{pmatrix}$$

Producen (s,s), que ya estaba

Producen (q,r)

56

AFNDs. Función de Transición extendida a palabras

Calculo de T*. Ejemplo 2:

- Se extiende la tabla de transición anterior para contener T*, insertando una nueva columna correspondiente a λ*

	a	b	~~λ~~	λ*
→*p	q			p
q	p,r,s	p,r	~~s~~	q,s,r
r		p,s	~~r,s~~	r,s
*s			~~r~~	r,s

57

AFNDs. Función de Transición extendida a palabras

Calculo de T*. Ejemplo 3:

- Y ahora se calcula la tabla de transición correspondiente a f'', cambiando las transiciones con a por λ*a λ * y las de b por λ *b λ *.

	a	b	~~λ~~	λ*
→*p	q			p
q	p,r,s	p,r	~~s~~	q,s,r
r		p,s	~~r,s~~	r,s
*s			~~r~~	r,s

→

	λ*aλ*	λ*bλ*
→*p	q,r,s	Φ
q	p,r,s	p,r,s
r	Φ	p,r,s
*s	Φ	p,r,s

58

AFND. Lenguaje aceptado por un AFND

- Una palabra $x \in \Sigma^*$ es aceptada por un AFND si:
  - f' (q0,x) y F tienen al menos un elemento común, es decir, que f'(q0,x) contiene al menos un estado final.

- El conjunto de todas las palabras aceptadas por un AFND es el lenguaje aceptado por ese AFND.

Formalmente:

$L_{AFND} = \{x / x \in \Sigma^* \text{ y } \exists\, q_o \to F\} = \{x / x \in \Sigma^* \text{ y } f'(q_o,x) \cap F \neq \emptyset\}$

59

AFND. Lenguaje aceptado por un AFND

- Al ser un AFND, desde q_o puede haber más de un camino para la palabra "***x***", y "***x***" **es aceptada sólo con que uno de los caminos lleve a un estado final**.

- Además:

  $\lambda \in$ L AFND si :

  - $q_o \in F$ ó
  - $\exists$ un estado final, $q \in F$, tal que está en

    relación T* con q_o (q_o T* q)

60

- Autómatas Finitos
  - Autómatas Finitos Deterministas (AFD)
    - AFD como reconocedores de lenguajes
    - Equivalencia y minimización de AFD
  - Autómatas Finitos No Deterministas (AFND)
- Equivalencia entre AFD y AFND

61

AFD equivalente a un AFND

- Dado un AFND siempre es posible encontrar un AFD que reconozca el mismo lenguaje:
  - El conjunto de los L_{AFND} = al conjunto de los L_{AFD}.
  - Un AFND no es más potente que un AFD, sino que un AFD es un caso particular de AFND.

- **Paso de AFND a AFD:**
  - Sea el AFND $A = (\Sigma, Q, f, q_0, F, T)$.
  - Se define a partir de A el AFD B, donde:

    $B = (\Sigma, Q', f^{\wedge}, q_0', F')$, tal que:

    Q' = P(Q) conjunto de las partes de Q que incluye a Q y a Ø.

    $q_0' = f'(q_0, \lambda)$ (f' extensión a palabra de f, i.e. todos los estados que tengan relación T* con q_0).

    $F' = \{C \mid C \in Q' \text{ y } \exists q \in C \mid q \in F\}$

    $f'(C,a) = \{C' \mid C' = \bigcup_{q \in C} f(q,a)\}$

62

AFD equivalente a un AFND

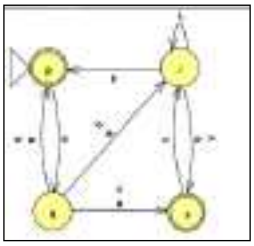
- EXPLICACIÓN **Paso de AFND a AFD:**
  1. Calculamos λ^* (usando T*)
  2. Calculamos $\lambda^* a\ \lambda^*\ \forall\ a \in \Sigma$
  3. lambda* a lambda* = a
  4. Transformamos los caminos múltiples en estados combinados:

     p, q, r pasa a ser {pqr}
  5. Construimos en AFD conexo considerando que las transiciones de {pqr} son las resultantes de hacer la unión de las transiciones de cada uno de los estados que lo forman.
  6. Será estado inicial q_0 o $q_0' = f'(q_o,\lambda)$ (f' extensión a palabra de f, i.e. todos los estados que tengan relación T* con q_0).
  7. Será estado final TODO estado combinado que contenga a un estado final o aquellos finales que estén conectados.

AFD equivalente a un AFND. Ejemplo

- Obtener el AFD correspondiente al siguiente AFND

	a	b	λ
→*p	q		
q	p,r,s	p,r	s
r		p,s	r,s
*s			r



- Pasos:
  1. Eliminar transiciones λ
     a) Determinar λ* (el cierre de las transiciones λ, T*)
     b) Obtener la tabla sin transiciones λ
  2. Aplicar algoritmo de creación de nuevos estados que pertenecen a P(Q), añadiendo su transiciones.

AFD equivalente a un AFND. Ejemplo

1. Eliminar transiciones λ
   a) Determinar λ* (el cierre de las transiciones λ) a partir de la tabla de transiciones.

	a	b	λ
→*p	q		
q	p,r,s	p,r	s
r		p,s	r,s
*s			r

↓

	a	b	~~λ~~	λ*
→*p	q			***p***
q	p,r,s	p,r	~~s~~	***q***,s,r
r		p,s	~~r,s~~	***r***,s
*s			~~r~~	***s***,r

AFD equivalente a un AFND. Ejemplo

1. Eliminar transiciones λ
   a) Determinar λ* (el cierre de las transiciones λ)

	a	b	~~λ~~	λ*
→*p	q			p
q	p,r,s	p,r	~~s~~	q,r,s
r		p,s	~~r,s~~	r,s
*s			~~r~~	r,s

   b) Obtener la tabla sin transiciones λ
   (transiciones con entrada λ*a λ*, para cada elemento, a, del alfabeto Σ)

	λ*aλ*	λ*bλ*
→*p	q,r,s	Ø
q	p,r,s	p,r,s
r	Ø	p,r,s
*s	Ø	p,r,s

AFD equivalente a un AFND. Ejemplo

2. Aplicar algoritmo de creación de nuevos estados que pertenecen a P(Q), añadiendo su transiciones.

	λ*aλ*	λ*bλ*
→*p	q,r,s	Ø
q	p,r,s	p,r,s
r	Ø	p,r,s
*s	Ø	p,r,s

	a	b
→*p	{q,r,s}	Ø
~~q~~	~~p,r,s~~	~~p,r,s~~
~~r~~	~~Ø~~	~~p,r,s~~
~~*s~~	~~Ø~~	~~p,r,s~~
{q,r,s}	**{p,r,s}U Ø U Ø**	**{p,r,s}U {p,r,s}U{p,r,s}**

	λ*aλ*	λ*bλ*
→*p	{q,r,s}	Ø
~~q~~	~~p,r,s~~	~~p,r,s~~
~~r~~	~~Ø~~	~~p,r,s~~
~~*s~~	~~Ø~~	~~p,r,s~~
{q,r,s}	{p,r,s}	{p,r,s}

AFD equivalente a un AFND. Ejemplo

2. Aplicar algoritmo de creación de nuevos estados que pertenecen a P(Q), añadiendo sus transiciones.

	λ*aλ*	λ*bλ*
→*p	q,r,s	Ø
q	p,r,s	p,r,s
r	Ø	p,r,s
*s	Ø	p,r,s

	λ*aλ*	λ*bλ*
→*p	{q,r,s}	Ø
~~q~~	~~p,r,s~~	~~p,r,s~~
~~r~~	~~Ø~~	~~p,r,s~~
~~*s~~	~~Ø~~	~~p,r,s~~
{q,r,s}	{p,r,s}	{p,r,s}

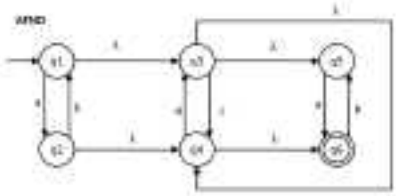
	λ*aλ*	λ*bλ*
→*p	{q,r,s}	Ø
~~q~~	~~{p,r,s}~~	~~{p,r,s}~~
~~r~~	~~Ø~~	~~{p,r,s}~~
~~*s~~	~~Ø~~	~~{p,r,s}~~
{q,r,s}	{p,r,s}	{p,r,s}

	λ*aλ*	λ*bλ*
→*p	{q,r,s}	Ø
~~q~~	~~p,r,s~~	~~p,r,s~~
~~r~~	~~Ø~~	~~p,r,s~~
~~*s~~	~~Ø~~	~~p,r,s~~
{q,r,s}	{p,r,s}	{p,r,s}
{p,r,s}	**{q,r,s} U Ø U Ø**	**Ø U {p,r,s} U {p,r,s}**

	λ*aλ*	λ*bλ*
→*p	{q,r,s}	Ø
{q,r,s}	{p,r,s}	{p,r,s}
{p,r,s}	**{q,r,s}**	**{p,r,s}**

Ejercicio 12 – Parte 3

Indica el grafo del Autómata Finito Determinista que se corresponde con el siguiente AFND:



	a	b	c	d	λ
->q1	q2				q3
q2		q1			q4
q3			q4		{q4, q5}
q4				q3	q6
q5	q6				
q6*		q5			

69

Ejercicio 12 – Parte 3

	a	b	c	d	λ	λ*	λ*a λ*	λ*b λ*	λ*c λ*	λ*d λ*
->q1	q2				q3	{q1,q3,q4,q5, q6}	{q2,q4,q6}	{q5}	{q4,q6}	{q3,q4,q5,q6}
q2		q1			q4	{q2,q4,q6}	--	{q1,q3,q4,q5,q6}	--	{q3,q4,q5,q6}
q3			q4		{q4, q5}	{q3,q4,q5,q6}	{q6}	{q5}	{q4,q6}	{q3,q4,q5,q6}
q4				q3	q6	{q4,q6}	--	{q5}	--	{q3,q4,q5,q6}
q5	q6					{q5}	{q6}	--	--	--
q6*		q5				{q6}	--	{q5}	--	--

70

Ejercicio 12 – Parte 3

	λ*	λ*a λ*	λ*b λ*	λ*c λ*	λ*d λ*
->q1	{q1,q3,q4,q5,q6}	{q2,q4,q6}	{q5}	{q4,q6}	{q3,q4,q5,q6}
q2	{q2,q4,q6}	--	{q1,q3,q4,q5, q6}	--	{q3,q4,q5,q6}
q3	{q3,q4,q5,q6}	{q6}	{q5}	{q4,q6}	{q3,q4,q5,q6}
q4	{q4,q6}	--	{q5}	--	{q3,q4,q5,q6}
q5	{q5}	{q6}	--	--	--
q6*	{q6}	--	{q5}	--	--

71

Ejercicio 12 – Parte 3

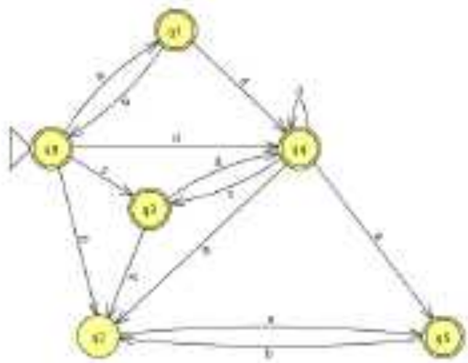
	a	b	c	d
-> {q1,q3,q4,q5,q6}*	{q2,q4,q6}	{q5}	{q4,q6}	{q3,q4,q5,q6}
{q2,q4,q6}*	ϕ	{q1,q3,q4,q5,q6}	ϕ	{q3,q4,q5,q6}
{q5}	{q6}	ϕ	ϕ	ϕ
{q4,q6}*	ϕ	{q5}	ϕ	{q3,q4,q5,q6}
{q3,q4,q5,q6}*	{q6}	{q5}	{q4,q6}	{q3,q4,q5,q6}
{q6}*	ϕ	{q5}	ϕ	ϕ
ϕ	ϕ	ϕ	ϕ	ϕ

72

Ejercicio 12 – Parte 3

Solución:

-> {q1,q3,q4,q5,q6}*=Q0
{q2,q4,q6}*=Q1
{q5}=Q2
{q4,q6}*=Q3
{q3,q4,q5,q6}*=Q4
{q6}*=Q5



Por simplicidad en el dibujo, el estado sumidero no se ha representado

73

3. Autómatas Finitos

Grado Ingeniería Informática

Teoría de Autómatas y Lenguajes Formales

Parte IV

TEMA 4. Gramáticas y Lenguajes Formales

4. Lenguajes y Gramáticas Formales

Grado Ingeniería Informática
Teoría de Autómatas y Lenguajes Formales

LENGUAJES FORMALES

2

- Definiciones
- Operaciones
- Reglas de derivación

3

Definiciones

Símbolo

Entidad abstracta, no se define (análogo al punto en geometría). Son letras, dígitos, caracteres, etc. También es posible encontrar símbolos formados por varios caracteres, pej: IF, THEN, ELSE, ...

Alfabeto (Σ)

Conjunto **finito no vacío** de letras o símbolos.

Sea "a" una letra y Σ un alfabeto, si a pertenece a ese alfabeto $\Rightarrow a \in \Sigma$

Ejemplos:

Σ_1= {A, B, C, ...,Z}

Σ_2= {0, 1}

Σ_3= {IF, THEN, ELSE, BEGIN, END}

4

Definiciones

Palabra, cadena, tira: toda secuencia finita de símbolos del alfabeto.

Ejemplos:

palabras sobre Σ_1 JUAN, ISABEL, etc.

palabras sobre Σ_2 00011101

palabras sobre Σ_3 IFTHENELSEEND

Notación: las palabras se representan por letras minúsculas del final del alfabeto

(x, y, z), pej x= JUAN, y= IFTHENELSEEND

5

Definiciones

Longitud de palabra

Es el número de símbolos que componen una palabra

La longitud de la palabra x se representa por $|x|$

Ejemplos:

$|x| = |\text{JUAN}| = 4$

$|y| = |\text{IFTHENELSEEND}| = 13$ (en Σ_1)

$|y| = |\text{IFTHENELSEEND}| = 4$ (en Σ_3)

Palabra vacía (λ)

Es aquella palabra cuya longitud es cero

Se representa por λ, $|\lambda| = 0$

Sobre cualquier alfabeto es posible construir λ

Utilidad: es elemento neutro en muchas operaciones con palabras y lenguajes

6

Definiciones

Universo del discurso, W(Σ): Es el conjunto de todas las palabras que se pueden formar con los símbolos de un alfabeto Σ

- ✓ También se denomina Lenguaje Universal de Σ
- ✓ Se representa como W(Σ)
- ✓ Es un conjunto infinito
- ✓ Ejemplo: sea $\Sigma_4 = \{A\}$, $W(\Sigma_4) = \{\lambda, A, AA, AAA, \ldots\}$ con un número ∞ de palabras

COROLARIO:

$\forall\ \Sigma,\ \lambda \in W(\Sigma) \Rightarrow$ La palabra vacía pertenece a todos los lenguajes universales de todos los alfabetos posibles

- Definiciones
- Operaciones
- Reglas de derivación

Operaciones con palabras

Operaciones con palabras sobre palabras de un universo del discurso dado

1 Concatenación de palabras

2 Potencia

3 Reflexión

1 Concatenación de palabras

sean dos palabras x, y tal que $x \in W(\Sigma)$, $y \in W(\Sigma)$, y sea

$$|x| = i = |x_1 x_2 \ldots x_i| \text{ e } |y| = j = |y_1 y_2 \ldots y_j|,$$

se llama **concatenación** de x con y, a:

$$x \cdot y = x_1 x_2 \ldots x_i\ y_1 y_2 \ldots y_i = \mathbf{z}, \text{ donde } z \in W(\Sigma)$$

1 Concatenación de palabras

Propiedades:

- ✓ Operación cerrada (o interna)
- ✓ Propiedad Asociativa
- ✓ Con elemento neutro.
- ✓ No conmutativa

Definiciones:

- ✓ Cabeza
- ✓ Cola
- ✓ Longitud de palabra

11

2 Potencia

Es la reducción de la concatenación a los casos que se refieren a una misma palabra

- ✓ potencia *i-ésima* de una palabra al resultado de concatenar esa palabra consigo misma i veces
- ✓ concatenación es asociativa $\Rightarrow$ no especificar el orden
- ✓ $x^i = x \cdot x \cdot x \,. \ldots x$ i veces
- ✓ $|x^i| = i \cdot |x|$
- ✓ se cumple:
  - $x^1 = x$
  - $x^{1+i} = x \cdot x^i = x^i \cdot x$ (i>0)
  - $x^{j+i} = x^j \cdot x^i = x^i \cdot x^j$ (i, j>0)
- ✓ Si se define $x^0 = \lambda$
  - (i, j>=0)

12

3 Reflexión de una palabra

Sea la palabra $x = a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_n$,

se denomina **palabra refleja** de x,

$x^{-1} = a_n \cdot a_{n-1} \cdot \ldots \cdot a_2 \cdot a_1$,

formada por los mismos símbolos en distinto orden

$|x^{-1}| = |x|$

13

Operaciones con lenguajes

¿Qué es un Lenguaje?

Se denomina **Lenguaje sobre el alfabeto Σ** a todo subconjunto del lenguaje universal de Σ ($L \subset W(\Sigma)$)

i.e. a todo conjunto de palabras sobre un determinado Σ

i.e. a todo conjunto de palabras generado a partir del alfabeto Σ

14

Operaciones con lenguajes

Son lenguajes especiales:

ϕ = Lenguaje vacío, $\phi \subset W(\Sigma)$

$\{\lambda\}$ = Lenguaje de la palabra vacía

ϕ y $\{\lambda\}$ se diferencian en el número de palabras (cardinalidad) que los forman $C(\phi) = 0$ mientras que $C(\{\lambda\})=1$ se parecen en que ϕ y $\{\lambda\}$ son lenguajes sobre cualquier alfabeto

Un alfabeto es uno de los lenguajes generados por el mismo:

$\Sigma \subset W(\Sigma)$, por ejemplo el chino

15

Operaciones con lenguajes

Operaciones con lenguajes sobre un un alfabeto dado Σ

1 Unión de lenguajes
2 Concatenación de lenguajes
3 **Binoide libre**
4 Potencia de un lenguaje
5 **Clausura o cierre positivo de un lenguaje**
6 **Iteración, clausura o cierre de un lenguaje**
7 **Reflexión de lenguajes**

16

1 Unión de lenguajes

Sean L1 y L2 definidos sobre el mismo alfabeto Σ, L1, L2 $\subset$ W(Σ), se llama **unión** de dos lenguajes, L1, L2 y se representa por L1$\cup$ L2 al lenguaje así definido:

$$L1 \cup L2 = \{ x \,/\, x \in L1 \text{ ó } x \in L2 \}$$

Es el conjunto formado indistintamente por palabras de uno u otro de los dos lenguajes (equivale a la suma)

$$L1 + L2 = L1 \cup L2$$

17

2 Concatenación de lenguajes

Sean L_1 y L_2 definidos sobre el mismo alfabeto, L_1, L_2 $\subset$ W(Σ), se llama **concatenación o producto** de dos lenguajes, L_1 y L_2 , y se representa por $L_1 \cdot L_2$ al lenguaje así definido:

$$L_1 \cdot L_2 = \{ xy \,/\, x \in L_1 \text{ AND } y \in L_2 \}$$

- ✓ Es el conjunto de palabras formado por la concatenación de palabras de L_1 con palabras de L_2
- ✓ Definición válida para lenguajes con algún elemento.
- ✓ Y con el lenguaje vacío: $\phi \cdot L = L \cdot \phi = \phi$

18

2 Concatenación de lenguajes

Propiedades

- ✓ Operación cerrada
- ✓ Propiedad Asociativa
- ✓ Con elemento neutro
- ✓ Propiedad distributiva respecto a la unión.

3 Binoide libre

- ✓ La concatenación (monoide) de lenguajes y la unión (monoide) de lenguajes constituyen un **binoide**
- ✓ Los símbolos de Σ se pueden considerar conjuntos de una sola palabra
- ✓ Con Σ, la unión y la concatenación se puede formar cualquier lenguaje sobre dicho Σ. Excepto ϕ y $\{\lambda\}$.

El alfabeto Σ es un conjunto de generadores para el conjunto L $\Rightarrow$ **L es el BINOIDE LIBRE (operaciones U y •) generado por Σ**

4 Potencia de un lenguaje

- Es la reducción de la concatenación a los casos que se refieren a un mismo lenguaje
- potencia *i-ésima* de un lenguaje al resultado de concatenar ese lenguaje consigo mismo *i* veces
- concatenación es asociativa ⇒ no especificar el orden
- $L^i = L \cdot L \cdot L \cdot \ldots \cdot L$; *i* veces
- Se define $L^1 = L$
- Se cumple:
  - $L^{1+i} = L \cdot L^i = L^i \cdot L$ (i>0)
  - $L^{j+i} = L^i \cdot L^j$ (i, j >0)
- Si se define $L^0 = \{ \lambda \}$, entonces (i≥0) (i, j ≥ 0)

5 Clausura o cierre positivo

Se representa como L^+ y es el lenguaje obtenido uniendo el lenguaje L con todas sus potencias posibles **excepto L^0**

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Ninguna clausura positiva contiene a λ, salvo si λ€L

Como Σ es un lenguaje sobre Σ, la clausura positiva de Σ será:

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i = W(\Sigma) - \{\lambda\}$$

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

6 Iteración, clausura o cierre

Se representa como L^* y es el lenguaje obtenido uniendo el lenguaje L con todas sus potencias posibles.

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

Toda clausura contiene a λ

* es el operador unario de Kleene

✓ Se cumple:

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

$$L^+ = L^* \cdot L = L \cdot L^*$$

✓ Como Σ es un lenguaje sobre Σ, se le puede aplicar el *:

$$\Sigma^* = W(\Sigma)$$ → El lenguaje universal es Σ^*

7 Reflexión de lenguaje

Se llama lenguaje reflejo o inverso de L y se representa por L^{-1} al lenguaje:

$$L^{-1} = \{x^{-1} \mid x \in L\}$$

es decir, al lenguaje formado por todas las palabras reflejas de L

- Definiciones
- Operaciones
- Reglas de derivación

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Producciones

Sea Σ un alfabeto

- ✓ Se llama producción a un **par ordenado** (x , y) donde x , $y \in \Sigma^*$

    $x \in \Sigma^*$, es la parte izquierda de la producción e

    $y \in \Sigma^*$, la parte derecha de la producción

- ✓ Se representa como: $x ::= y$

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Derivación directa

- ✓ Sea Σ un alfabeto
- ✓ Sea (*x, y*) una producción sobre palabras de Σ, x::=y
- ✓ Sean v y w dos palabras sobre Σ (i.e v, w ∈ Σ*)

Se dice

"w es derivación directa de v" ó
"v produce directamente w" $v \to w$
"w se reduce directamente a v"

si ∃ dos palabras z , u ∈ Σ* tales que v = z·x·u y w = z·y·u

COROLARIO

Si x ::= y es una producción sobre Σ ⇒ $x \to y$
(una regla de escritura es una derivación directa)

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

27

Derivación directa

Ejemplos

- ✓ Sea Σ el alfabeto castellano de las letras mayúsculas y ME::=BA una producción sobre Σ

  CABALLO es derivación directa de CAMELLO
  (CAMELLO produce directamente CABALLO)

- ✓ Y con la producción CA::=PE sobre Σ

  PERA es derivación directa de CARA

En el castellano no son así las cosas, existen las raíces.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

28

Derivación directa

En un conjunto de producciones

- ✓ Sea Σ un alfabeto y P un conjunto de producciones sobre Σ
- ✓ Sean v y w dos palabras sobre Σ, v, w $\in\Sigma^*$

Se dice que "w es derivación directa de v" / "v produce directamente w" / "w se reduce directamente a v" $\}$ $v \rightarrow w$

si $\exists$ dos palabras z, u $\in\Sigma^*$ tales que $v = z{\cdot}x{\cdot}u$ y $w = z{\cdot}y{\cdot}u$ y se cumple **(x::=y)** $\in$ **P**

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Derivación

- ✓ Sea Σ un alfabeto y P un conjunto de producciones sobre Σ
- ✓ Sean *v* y *w* dos palabras sobre Σ, *v*, *w* $\in\Sigma^*$

Se dice que "w es derivación de v" / "v produce w" / "w se reduce a v" $\}$ $v \xrightarrow{+} w$

si $\exists$ una secuencia finita de palabras, $u_0, u_1, u_2, \ldots u_n$ (n>0) $\in\Sigma^*$

tales que $v = u_0$

$$\left.\begin{array}{l} u_0 \rightarrow u_1 \\ u_1 \rightarrow u_2 \\ \ldots\ldots \\ u_{n-1} \rightarrow u_n \\ w = u_n \end{array}\right\}$$ **Derivación de longitud n**

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Relación de Thue

- ✓ Sea Σ un alfabeto y P un conjunto de producciones sobre Σ
- ✓ Sean v y w dos palabras sobre Σ, es decir v, w ∈Σ*

Se dice que existe una relación de Thue entre v y w y se representa por **v *→ w** si se verifica que:

v +→ w ó
v = w

o ∃ una derivación de longitud *n* o son iguales

Propiedades

- ✓ Reflexiva
- ✓ Transitiva
- ✓ Simétrica (en general NO se cumple)

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

GRAMÁTICAS FORMALES

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

- Definiciones
- Jerarquía de Chomsky
- Árboles de derivación

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

¿Qué es una gramática?

Una gramática describe la **estructura** de las frases y de las palabras de un lenguaje y se aplica por igual a:

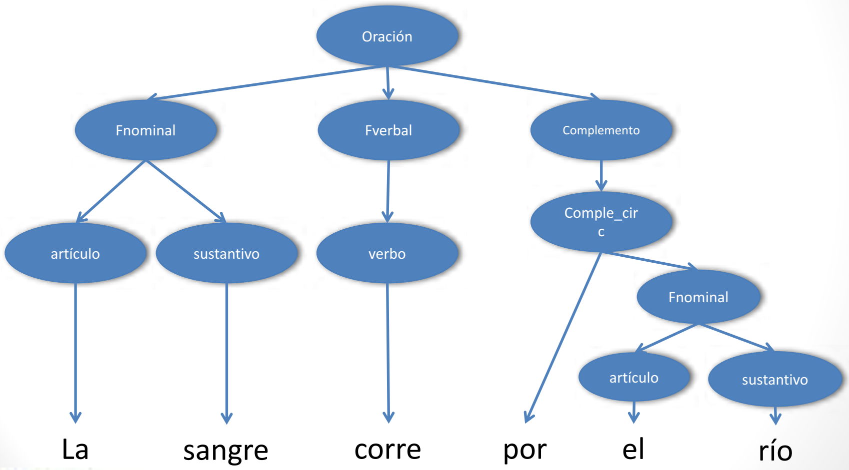
- ✓ Las lenguas naturales humanas
- ✓ Lenguajes de programación.

Una gramática es un **"ente formal"** para especificar de manera finita el conjunto de cadenas de símbolos que constituyen un lenguaje.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramática del Castellano

Una gramática del castellano como diagrama sintáctico



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Reglas de producción

Una gramática del castellano en notación de Backus

<Oración>::= <Fnominal> <Fverbal> | <Fnominal> <Fverbal><Complemento>

<Fnominal>::= <Sustantivo> | <NomPr> | <Artículo> <Sustantivo>

| <Artículo> <Sustantivo> <Adjetivo>

| <Artículo> <Sustantivo> <Adjetivo>

| < Fnominal >de < Fnominal >

<Fverbal>::= <Verb> | <Verb> <Adverbio>

<Complemento>::=<ComDir> |<ComIn> |<ComCir> | <ComDir> <ComIn> <ComCir>

<ComDir>::= <Fnominal> , <ComIn>::= a <Fnominal> | para <Fnominal> | ....

<ComCir> ::= en <Fnominal> | desde <Fnominal> | cuando <Fnominal> | ....

Donde: <Sustantivo>, <Adjetivo>, <Adverbio>, <Artículo>, <NomPr>, <Verbo>, etc toman como valores palabras propias de estas categorías gramaticales

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Definición

Cómo se determina (define) una gramática:
Es una cuádrupla: **(Σ_T, Σ_N, S, P)**, Σ_T y Σ_N son alfabetos:

- **Σ_T: Alfabeto de símbolos terminales.** Todas las cadenas del lenguaje representado por la G (L(G)) están formadas con símbolos de este alfabeto.
- **Σ_N: Alfabeto de símbolos No Terminales.** Conjunto de símbolos auxiliares introducidos como elementos auxiliares para la definición de G pero que no figuran en las cadenas de L(G).
- **S: Axioma** o símbolo destacado. Es un símbolo NT a partir del que se comienzan a aplicar las reglas de P.
- **P:** conjunto de reglas de producción: **u::=v** donde $u \in \Sigma^+$ y $v \in \Sigma^*$
  u = xAy tal que x, y $\in \Sigma^*$ y A $\in \Sigma_N$.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Definición

Σ_T: **Alfabeto de símbolos terminales**

Σ_N: **Alfabeto de símbolos NO terminales**

Axioma

G = ({0,1}, {N,C}, N, P)

P = {N::= NC,
N::=C,
C::= 0,
C ::=1}

Producciones

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Definición

Se cumple entre los alfabetos de G:

$\Sigma_T \cup \Sigma_N = \Sigma$ Alfabeto o vocabulario de G

$\Sigma_T \cap \Sigma_N = \phi$ (son disjuntos)

Notación de Backus: Si u::=v y u::= w son dos reglas de producción de P, entonces se puede escribir u ::= v | w

Se denomina notación **BNF**: *Forma Normal de Backus o Forma Normal de Backus-Naur*

Ejemplo:

sea G = ({0,1,2,3,4,5,6,7,8,9}, {N,D}, N,P)

P = { N::= ND | D ; D::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

Forma sentencial

Sea una *G*, sea $x \in \Sigma^*$, es decir sea $x \in (\Sigma_T \cup \Sigma_N)^*$,

x se denomina **forma sentencial de G** si se verifica:

$$S \xrightarrow{*} x$$

es decir, que existe una ***relación de Thue*** entre el axioma y *x*.

Si **x** $\in \Sigma_T^*$ se dice que **x** es una **Sentencia o instrucción del lenguaje descrito por G**

Lenguaje asociado a una gramática

Sea la gramática:

$G_1 = (\Sigma_T, \Sigma_N, S, P)$

Se llama:

lenguaje asociado a G1
lenguaje generado por G1
o lenguaje descrito por G1

al **conjunto de todas las sentencias (palabras) generadas** por G1, es decir:

$$L(G_1) = \{x \mid S \xrightarrow{*} x,\ x \in \Sigma^*_T\}$$

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Recursividad

Sea *G*,

- ✓ Una *G* se llama **recursiva en *U***, $U \in \Sigma_{NT}$, si se cumple:

  $$U \xrightarrow{+} x\,U\,y$$

  - Si $x = \lambda$ ($U \xrightarrow{+} U\,y$) se dice que *G* es **recursiva a izquierdas**
  - Si $y = \lambda$ ($U \xrightarrow{+} x\,U$) se dice que G es **recursiva a derechas**
- ✓ Una regla de producción es recursiva si tiene la forma:

  $$U ::= x\,U\,y$$
- ✓ Si un lenguaje es infinito, la gramática que lo representa tiene que ser recursiva

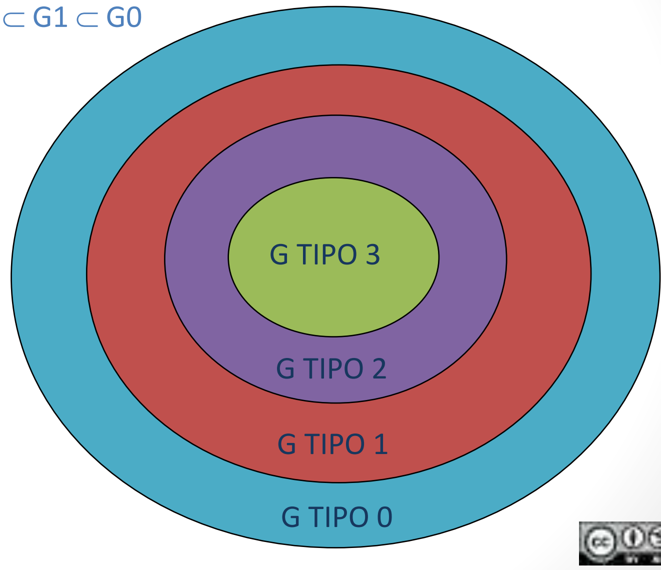
A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

- Definiciones
- Jerarquía de Chomsky
- Árboles de derivación

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

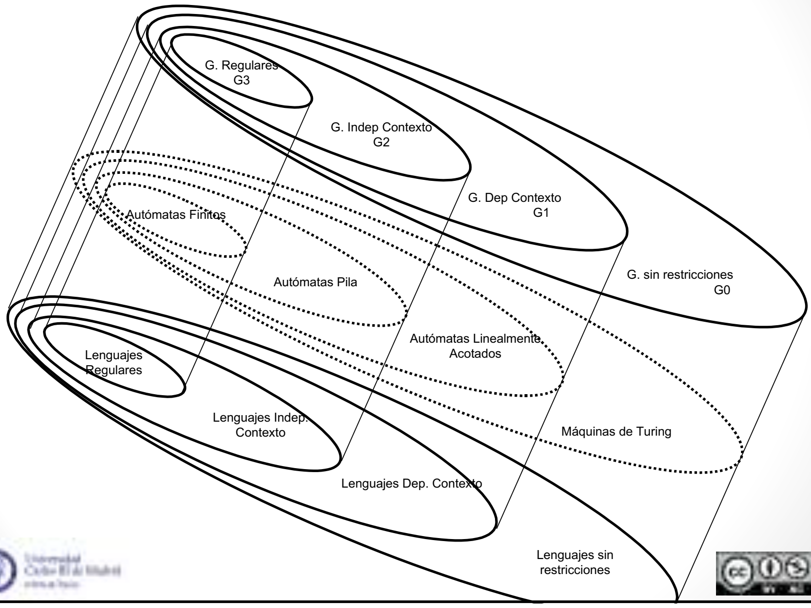
Jerarquía de Chomsky

$G3 \subset G2 \subset G1 \subset G0$



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Máquinas, Lenguajes y Algoritmos



45

Tipo 0

G= (Σ_T, Σ_N, S, P)
$\Sigma_T \cup \Sigma_N = \Sigma$, alfabeto gramática,
$\Sigma_T \cap \Sigma_N = \phi$

G0
No Restringidas
o con Estructura de Frase

u ::= v $u \in \Sigma^+$ $v \in \Sigma^*$

Única restricción: **λ::= v $\notin$P**

Forma sentencial:
u = xAy, x,y $\in \Sigma^*$, A $\in \Sigma_N$

Ejemplo:
G= {(0,1), (S,A), S, P}, donde:
P={S→0S0A, SA→AS, 0A→1}

Con estructura de frases:
(xAy ::= xvy) $\in$ P, donde:
x,y $\in \Sigma^*$, A $\in \Sigma_N$, v $\in \Sigma^*$
En xAy ::= xvy cuando v=λ,
xAy ::= xy reglas compresoras

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

46

Tipo 0

- ✓ Los lenguajes que son representados por G0 se llaman lenguajes sin restricciones
- ✓ Chomsky 1959. Todo L(G0) puede ser descrito por una G0 con estructura de frases.

Ejemplo

G= ({a,b},{A,B.C},A,P)
P = {A::= aABC / abC
CB::= BC
bB::=bb
bC::=b}

G'= {a,b},{A,B.C,X,Y},A,P')
P' = {A::= aABC / abC
CB::=XB, XB::=XY
XY::=BY, BY::=BC
bB::=bb
bC::=b}

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

47

Tipo 1

G = (Σ_T, Σ_N, S, P)
$\Sigma_T \cup \Sigma_N = \Sigma$, alfabeto gramática,
$\Sigma_T \cap \Sigma_N = \phi$

G1
Sensibles al Contexto

xAy ::= xvy

$x,y \in \Sigma^*$, $A \in \Sigma_N$
$v \in \Sigma^+$ → No permite reglas compresoras
Excepción: ($S::= \lambda$) $\in$ P

Ejemplo:
G= ({a,b,c}, {A,B},A, P},
P={A ::= Aaa | a,
aA ::= aB | a,
aCCc ::= aAaCc,
C ::= c}

Los Lenguajes representados por una gramática de Tipo 1 se llaman ***lenguajes dependientes del contexto*** o lenguajes sensibles al contexto (se puede cambiar A por v, siempre en el contexto x...y)

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

48

Tipo 1

Los lenguajes que son representados por G1 se llaman **Lenguajes sensibles al contexto**

donde

$$\lambda \in L(G1) \text{ Sii } (S::=\lambda) \in P$$

Ejemplo de G que no es G1

G= ({a,b}, {S}, S, P) ,

P= { S::= aaaaSbbbb,
aSb::= ab}

Ejemplo de G que si es G1

G= ({a,b}, {S}, S, P) ,

P = { S::= aaaaSbbbb,
aSb::= abb}

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

49

Tipo 1. No decrecimiento

Propiedad de NO DECRECIMIENTO de las G1

Las cadenas que se obtienen en cualquier derivación de una G1 son de longitud no decreciente, es decir:

$$\mathbf{u \rightarrow v \Rightarrow |v| \geq |u|}$$

La longitud de la parte derecha de la producción es mayor o igual a la longitud de la parte izquierda

Demostración

$$\alpha A\beta \rightarrow \alpha\gamma\beta$$

donde $\gamma \in (\Sigma_T \cup \Sigma_{NT})^+$ por definición de G1 (no compresoras)

es decir, $\gamma \neq \lambda$ siempre, lo que implica $|\gamma| \geq 1$

y como $|A| = 1$ como mínimo, queda demostrado

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

50

Tipo 2

$G = (\Sigma_T, \Sigma_N, S, P)$
$\Sigma_T \cup \Sigma_N = \Sigma$, alfabeto gramática,
$\Sigma_T \cap \Sigma_N = \phi$

G2
De Contexto Libre

A ::= v

$v \in \Sigma^*$
$A \in \Sigma_N$

$r \in P$ se caracterizan por tener **un sólo símbolo NT** en su parte izquierda

$(S ::= \lambda) \in P$ y $(A ::= \lambda) \notin P$
(algoritmo limpieza reglas NO generativas)

Ejemplo:
G= ({0,1,2,3,4,5,6,7,8,9},{N,D},N, P)
P={N → DN | D,
D → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

Contexto Libre: Se puede cambiar **A** por **v**, en cualquier contexto

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Tipo 2

Los lenguajes que son representados por G2 se llaman **Lenguajes no sensibles al contexto o de contexto libre**

$\forall$ L(G2) $\exists$ L(G2') sin reglas $A ::= \lambda$ $(A \neq S)$

Si $\lambda \in L(G2)$ → **puede aplicarse el algoritmo de eliminación reglas NO generativas.**

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Tipo 3

$G= (\Sigma_T, \Sigma_N, S, P)$
$\Sigma_T \cup \Sigma_N = \Sigma$, alfabeto gramática,
$\Sigma_T \cap \Sigma_N = \phi$

G3
Gramáticas Regulares

G3 Lineales por la Izda.
A::= a
A::= V a
S::= λ

G3 Lineales por la Drcha.
A::= a
A::= a V
S::= λ

$a \in \Sigma_T$
$A,V \in \Sigma_N$
S es axioma

Ejemplo:
G= {(a,b), (S,A),S, P},
P={(S $\rightarrow$ aS, S $\rightarrow$ aA, A$\rightarrow$ bA, A $\rightarrow$ b)}

r $\in$ P **un sólo símbolo NT** en su parte izda y su parte dcha comienza por un T seguido o no de NT (al revés en lineal derecha)

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Tipo 3

Los lenguajes que son representados por G3 se llaman **lenguajes regulares**

$\forall$ L(G3) $\exists$ L(G3') sin reglas A::= λ (A $\neq$ S)

Si $\lambda \in$ L(G3).

Ver algoritmo eliminación reglas NO generativas.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Jerarquía de Chomsky. Resumen

$G3 \subset G2 \subset G1 \subset G0$

G0. No Restringidas o con Estructura de Frase

u ::= v $u \in \Sigma^+$ $v \in \Sigma^*$

Única restricción: **λ::= v $\notin$P**

Con Estructura De Frases:
(xAy::=xvy) $\in$ P, donde: x, y $\in \Sigma^*$, A $\in \Sigma_N$, v $\in \Sigma^*$
Si v=λ → xAy ::= xy reglas compresoras

G1. Sensibles al Contexto

xAy ::= xvy

x, y $\in \Sigma^*$, A$\in \Sigma_N$

v $\in \Sigma^+$ → No permite reglas **compresoras**
Excepción: **(S::= λ) $\in$ P**

G2. De Contexto Libre

A ::= v

v $\in \Sigma^*$
A $\in \Sigma_N$

(S::= λ) $\in$P y **(A::= λ) $\notin$P**

G3. Gramáticas Regulares

G3 Lineales por la Izda.
A::= a
A::= V a
S::= λ

G3 Lineales por la Drcha.
A::= a
A::= a V
S::= λ

a $\in \Sigma_T$
A,V $\in \Sigma_N$

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

55

Gramáticas equivalentes

Dos gramáticas son equivalentes si representan el mismo lenguaje.

Dada una gramática lineal por la derecha cualquiera, existe otra lineal por la izquierda equivalente y viceversa.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

56

Gramáticas equivalentes

ALGORITMO: 3 PASOS.

PASO 1.

Construir una gramática equivalente que no sea recursiva en el axioma (axioma inducido):

1. se añade un nuevo símbolo en el alfabeto Σ_N, B
2. $\forall$ S::= x, donde x $\in \Sigma^+$, se añade una regla B::= x
3. Se transforman las reglas A::= a S (que desaparecen) en reglas del tipo A::= a B.
4. Las reglas tipo S::= λ no se ven afectadas por este algoritmo.

Nota: las reglas S ::= x, x $\in \Sigma^+$, no desaparecen

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

57

Gramáticas equivalentes

ALGORITMO: 3 PASOS.

PASO 1. Quitar el axioma inducido:

Construir una gramática equivalente que no sea recursiva en el axioma:

G1 = ({a,b}, {S, A}, S, P)

P = { S ::= bA,
A ::= aS | a }

PASO 1 →

G2 = ({a,b}, {S, A, B}, S, P)

P = { S ::= bA,
A ::= aB | a
B ::= bA}

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

58

Gramáticas equivalentes

ALGORITMO: 3 PASOS.

PASO 2.

Construcción de un grafo dirigido a partir de la gramática.

a) número de nodos = $C(\Sigma_N) + 1$, cada nodo etiquetado con símbolos de Σ_N y otro con λ

b) cada $A ::= a\ B \in P$ $\longrightarrow$ $A \xrightarrow{a} B$

c) cada $A ::= a \in P$ $\longrightarrow$ $A \xrightarrow{a} \lambda$

d) si $S ::= \lambda \in P$ $\longrightarrow$ $S \longrightarrow \lambda$

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas equivalentes

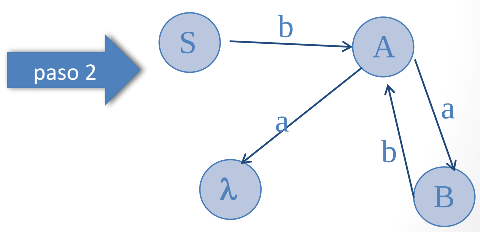
ALGORITMO: 3 PASOS.

PASO 2.

Construcción de un grafo dirigido a partir de la gramática.

G2 = ({a,b}, {S, A, B}, S, P}

P = { S ::= bA,
A ::= aB | a
B ::= bA}



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas equivalentes

ALGORITMO: 3 PASOS.

PASO 3.

Se intercambian las etiquetas y se construye un nuevo grafo.

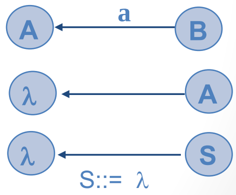
Transformación del grafo dirigido anterior:

a) intercambiar etiquetas de S y λ

b) invertir sentido de todos arcos

c) deshacer el camino y generar las nuevas reglas

⇒ interpretar el grafo para obtener la G3LI equivalente



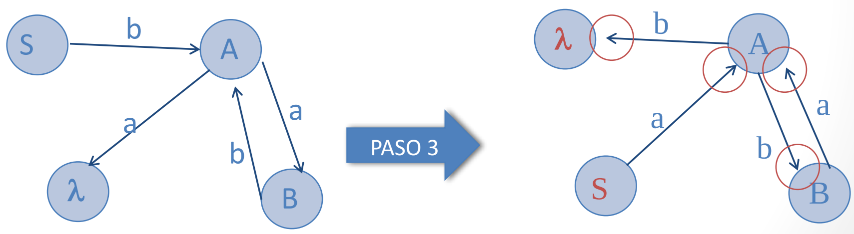
A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas equivalentes

ALGORITMO: 3 PASOS.

PASO 3.

Se intercambian las etiquetas y se construye un nuevo grafo.

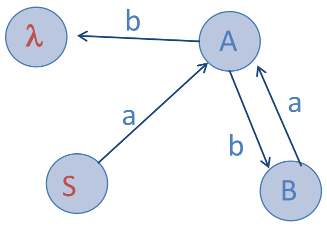


A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas equivalentes

ALGORITMO: 3 PASOS.

A partir del grafo, se obtiene la gramática equivalente a izquierdas.



G3 = ({a,b}, {S, A, B}, S, P}

P = { S ::= Aa,
A ::= Bb | b
B ::= Aa}

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas equivalentes

ALGORITMO: 3 PASOS.

Comprobemos que las gramáticas son equivalentes...

G1 = ({a,b}, {S, A}, S, P}

P = { S ::= bA,
A ::= aS | a }

G3 = ({a,b}, {S, A, B}, S, P}

P = { S ::= Aa,
A ::= Bb | b
B ::= Aa}

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

- Definiciones
- Jerarquía de Chomsky
- Árboles de derivación

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Árboles de derivación

- ✓ A las derivaciones de las *G tipo 1, 2 y 3* les corresponde un árbol de derivación equivalente, también llamado

  “árbol sintáctico” ó
  “parse tree”

- ✓ Representa las producciones aplicadas durante la generación de **una sentencia**, es decir, su estructura de acuerdo con la G

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

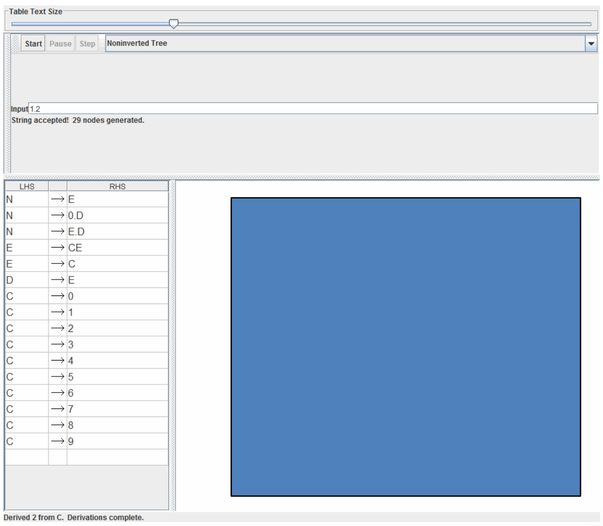
Árboles de derivación

Es un árbol ***ordenado*** y ***etiquetado*** que se construye:

- ✓ La raíz se denota por el axioma de G (S habitualmente)
- ✓ Una **derivación directa** se representa por un conjunto de ***ramas que salen de un nodo dado (parte izquierda de la P)***
- ✓ Aplicar una regla un símbolo de la parte izq. queda sustituido por una palabra *u* de la parte dcha. Por cada uno de los símbolos de *u* se dibuja una rama que sale del NT a ese símbolo:

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Árboles de derivación



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Árboles de derivación

En una G1, además, se debe conservar el contexto. Para cada rama:

- ✓ el nodo de partida se llama **padre** del nodo final
- ✓ el nodo final es **hijo** del nodo padre
- ✓ dos nodos hijos del mismo padre se llaman **hermanos**
- ✓ un nodo es **ascendente** de otro si es su padre o ascendiente de su padre
- ✓ un nodo es **descendiente** de otro si es su hijo o descendiente de sus hijos

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

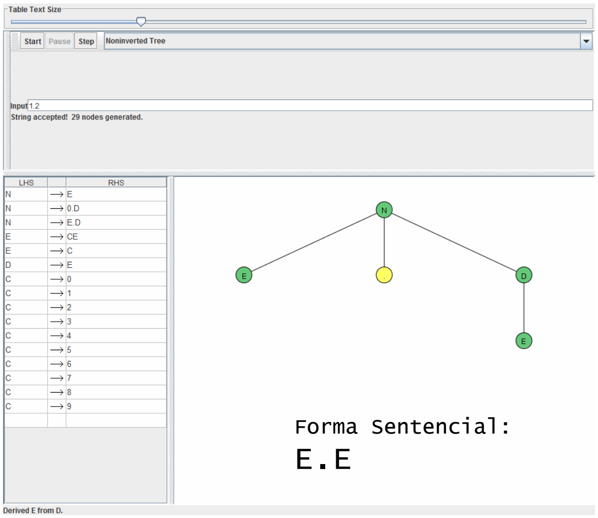
Árboles de derivación

- ✓ A lo largo del proceso de construcción del árbol, los nodos finales de cada paso sucesivo, leídos de izqda. a dcha. dan la ***forma sentencial*** obtenida por la derivación representada por el árbol.

- ✓ El ***conjunto de las hojas del árbol*** (nodos denotados por símbolos terminales o λ) leídos de izqda. a dcha. nos dan la ***sentencia*** generada por la derivación

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

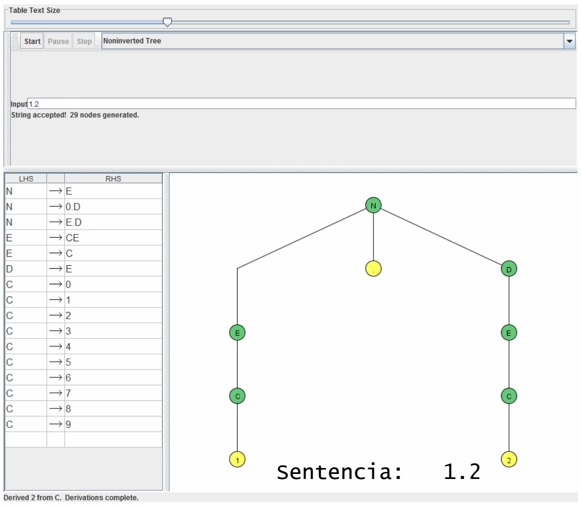
Árboles de derivación



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

71

Árboles de derivación



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

72

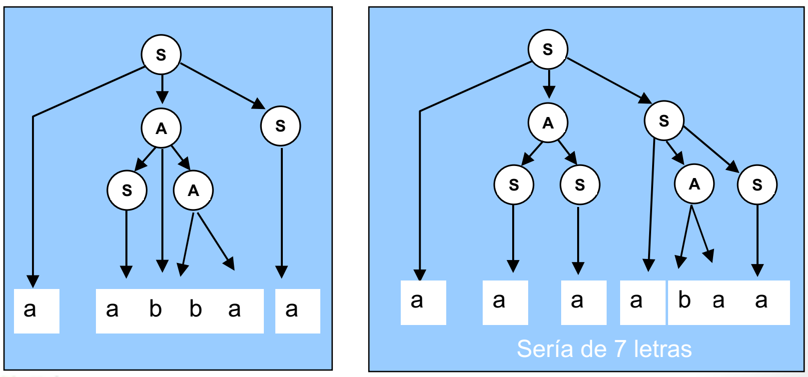
Árboles de derivación

Dada la G = ({a,b}, {A,S}, S, {S::= aAS | a, A::= SbA | SS | ba}). Hallar un árbol de derivación para una sentencia de 6 letras.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Árboles de derivación

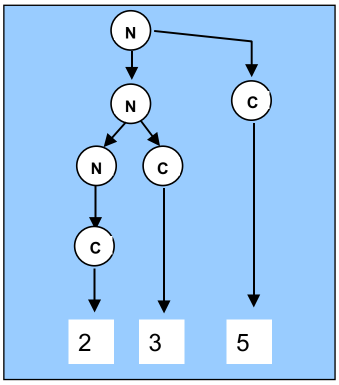
Dada la G = ({a,b}, {A,S}, S, {S::= aAS | a, A::= SbA | SS | ba}). Hallar un árbol de derivación para una sentencia de 6 letras.



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Árboles de derivación

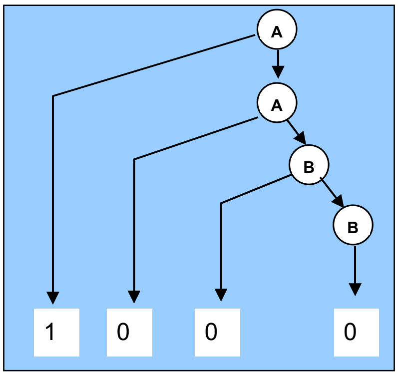
Dada la G = ({0,1,2,3,4,5,6,7,8,9}, {N,C}, N, {N ::= NC | C, C ::= 0|1|2|3|4|5|6|7|8|9}). Hallar un árbol de derivación para N → 235



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Árboles de derivación

Dada la G = ({0,1}, {A,B}, A, {A::= 1A | 0B, B::= 0B | 0}) una de cuyas derivaciones válidas es: A → 1A → 10B → 100B → 1000. Hallar un árbol de derivación para A → 1000

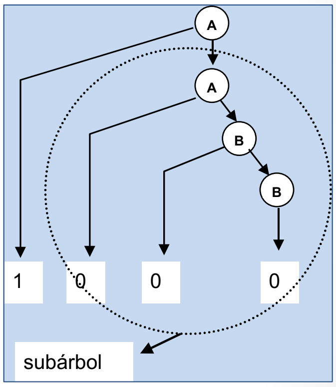


A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Subárboles de derivación

Dado un árbol A correspondiente a una derivación, se llama **subárbol de A** al árbol

- ✓ cuya raíz es un nodo cualquiera de A,
- ✓ cuyos nodos son todos los descendientes de la raíz del subárbol en A
- ✓ y cuyas ramas son todas las que unen dichos nodos entre si en A.

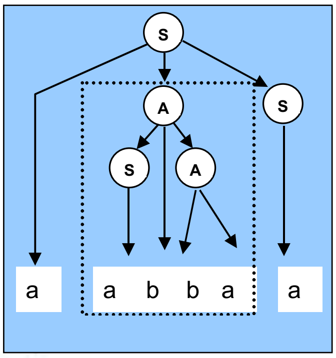


A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Subárboles de derivación

Teorema

Las hojas de un subárbol, leídas de izda. a dcha., forman una frase respecto al símbolo NT raíz del subárbol



abba es una frase de la forma sentencial **aabbaa** respecto del **símbolo A**

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

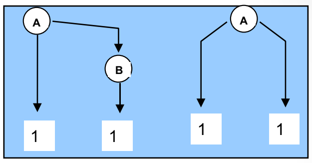
Ambigüedad

- ✓ Concepto relacionado con el de árbol de derivación:
- ✓ Si una sentencia puede obtenerse en una G por medio de dos o más árboles de derivación diferentes, **la sentencia es ambigua**
- ✓ Una **G** es **ambigua** si contiene **al menos una sentencia ambigua**

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Ambigüedad

Existen 3 niveles de ambigüedad:



- ✓ ***Sentencia***: una sentencia es ambigua si puede obtenerse por medio de dos o más árboles de derivación diferentes

  ej: G = ({1}, {A,B}, A, {A::= 1B | 11, B::= 1})

- ✓ ***Gramática**:* es ambigua si contiene al menos una sentencia ambigua, ej: la G anterior
- ✓ ***Lenguaje inherentemente ambiguo***: si todas las gramáticas que lo generan son ambiguas.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Ambigüedad

- Aunque una G sea ambigua, es posible que el lenguaje que describe no sea ambiguo [Floyd 1962] ⇒ es posible encontrar una G equivalente que no lo sea
- Existen lenguajes para los que NO es posible encontrar G no ambiguas ⇒ Lenguajes Inherentemente Ambiguos [Gross 1964]
- La propiedad de ambigüedad es indecidible. Tan solo es posible encontrar condiciones suficientes que aseguren que una G es no ambigua
- **Indecidible**: no existe un algoritmo que acepte una G y determine con certeza y en un tiempo finito si una G es ambigua o no.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Ambigüedad

Lenguajes Inherentemente Ambiguos: para los que NO es posible encontrar G no ambiguas

Ejemplo

$L = \{\{a^n b^m c^m d^n\} \cup \{a^n b^n c^m d^m\} \ / \ m,n \geq 1\}$

Ejemplo

L = {11} NO es inherentemente ambiguo

G = ({1}, {A,B}, A, {A::= 1B / 11, B::= 1})

G' = ({1}, {A}, A, {A::=11}) --> Gramática NO ambigua

L(G)=L(G')

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Ambigüedad

Dada la G = ({a,b}, {A,B,S}, S, P)

P = {

S::= bA | aB

A::= bAA | a | aS

B ::= b |BS | aBB}

demostrar que es una G ambigua al serlo la sentencia "**aabbab**"

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

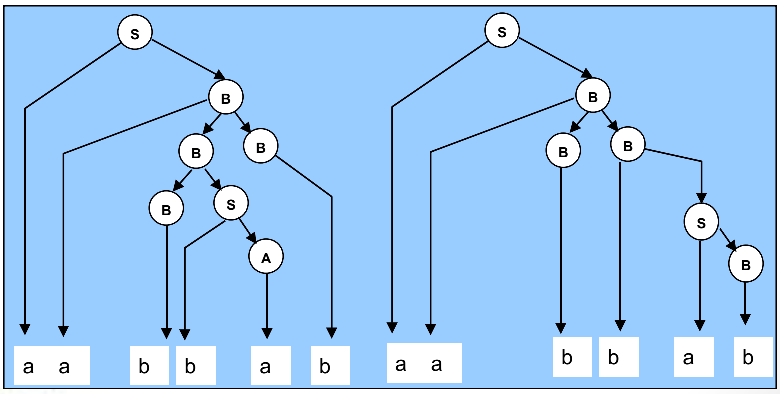
Ambigüedad

Dada la G = ({a,b}, {A,B,S}, S, P)

P = {

S::= bA | aB, A::= bAA | a | aS, B ::= b | BS | aBB}

demostrar que es una G ambigua al serlo la sentencia "**aabbab**"



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Segunda parte

GRAMÁTICAS FORMALES

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

- Gramáticas Tipo 2
- Gramáticas bien formadas
- Formas normales

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Independientes del Contexto

Son las gramáticas de tipo 2 en la jerarquía de Chomsky.

Relevancia

- ✓ son las empleadas en la Definición de Lenguajes de Programación y en la compilación de los mismos

Compilador
- analizador → aceptador de la G
  - explorador (AF)
  - reconocedor (AP)
- generador de código → Traductor

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Independientes del Contexto

A los lenguajes generados por gramáticas del tipo 2 de la jerarquía de Chomsky se les denomina **Lenguajes independientes del contexto o lenguajes de contexto libre**

- ✓ Se representan como L(G2)
- ✓ Existen algoritmos que permiten reconocer si un L(G2) es vacío, finito o infinito

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Independientes del Contexto

1. Dada una G, el Lenguaje que genera, ¿es vacío o no?:

Sea **G2**, m = $C(\Sigma_{NT})$, **L(G2)** $\neq \phi$ si $\exists$ x $\in$ L(G2) tal que x puede generarse con un árbol de derivación en el que todos los caminos tienen longitud $\leq$ m

Se generan todos los árboles de derivación con caminos $\leq$ m = $C(\Sigma_{NT})$ mediante el algoritmo:

a. conjunto de árboles con longitud 0 (un árbol con S como raíz y sin ramas)

b. a partir del conjunto de árboles de longitud n, generamos el conjunto de longitud n+1 < m +1 aplicando al conjunto de partida una producción que no haga duplicarse algún NT en el camino considerado

c. se aplica el paso b) recursivamente hasta que no puedan generarse más árboles con caminos de longitud $\leq$ m. Al ser m y el número de reglas de P finito $\Rightarrow$ el algoritmo termina

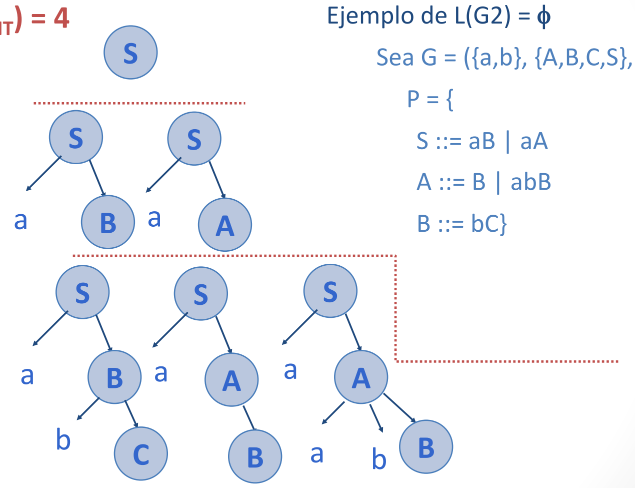
L(G2) = ϕ si ninguno de los árboles genera una sentencia

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Independientes del Contexto

m=$C(\Sigma_{NT})$ = 4

1. m=0
2. m=1
3. m=2
4. m=3



No se generan sentencias y salen NT ya obtenidos

Lenguaje vacío

Ejemplo de L(G2) = ϕ

Sea G = ({a,b}, {A,B,C,S}, S, P)

P = {

S ::= aB | aA

A ::= B | abB

B ::= bC}

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Independientes del Contexto

2. Si L(G2) es no vacío, comprobar si L(G2) = ∞

✓ Se construye un grafo cuyos nodos están etiquetados con los símbolos de (Σ_{NT}) mediante el algoritmo:

a) si $\exists$ una producción A::= α B β, se crea un arco de A a B donde A,B $\in \Sigma_{NT}$ y $\alpha,\beta \in \Sigma^*$

b) si no existen ciclos en el grafo el L(G2) = finito

c) L(G2) = ∞ si existen ciclos accesibles desde el axioma que corresponden a derivaciones de la forma A $\rightarrow^+$ α A β, donde $|\alpha|+|\beta|>0$ (que no sean λ las dos a la vez).

L(G2) $\neq \infty$ si no hay ciclos en el grafo

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Independientes del Contexto

Ejemplo de **L(G2) = ∞**

Sea G = ({a,b,c}, {A,B,C,S}, S, P)

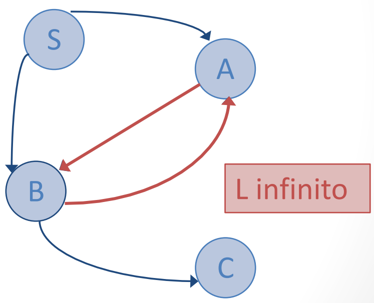
P = {

S ::= aB / aA

A ::= abB

B ::= bC / aA

C ::= c }



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

- Gramáticas Tipo 2
- Gramáticas bien formadas
- Formas normales

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

Transformación de una G dada en otra equivalente cuyas reglas de producción estén en un formato carente de imperfecciones:

1. **Limpieza de Gramáticas**
   a. Reglas Innecesarias
   b. Símbolos Inaccesibles
   c. Reglas Supérfluas
2. **Eliminación de símbolos no generativos**
3. **Eliminación de reglas no generativas**
4. **Eliminación de reglas de redenominación**

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

Transformación de una G dada en otra equivalente cuyas reglas de producción estén en un formato carente de imperfecciones:

1. **Limpieza de Gramáticas**
   a. Reglas Innecesarias
   b. Símbolos Inaccesibles
   c. Reglas Superfluas
2. **Eliminación de símbolos no generativos**
3. **Eliminación de reglas no generativas**
4. **Eliminación de reglas de redenominación**

Gramática Reducida

Gramática Limpia

Gramática Bien Formada

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

1. **Limpieza de Gramáticas**
   a. **Reglas Innecesarias**: las reglas **A::= A** $\in$**P** son innecesarias y hacen que G sea ambigua $\Rightarrow$ eliminarlas

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

1. **Limpieza de Gramáticas**
   b. **Símbolos Inaccesibles:** sea U::= x $\in$P, donde U$\in\Sigma_N \neq$ S y no aparece en la parte derecha de ninguna otra regla de producción, se dice que U es inaccesible.

   Todo símbolo U$\in\Sigma_N$ no inaccesible debe cumplir S $*\rightarrow$ xUy.

   Eliminación de símbolos inaccesibles:

   1. Hacer una lista con todos los símbolos de la gramática (T y NT)
   2. Marcar el axioma de la gramática.
   3. Dado *xUy ::= xuy* → Marcar todos los símbolos que aparecen en la cadena *u* de la parte derecha.
   4. Si en el paso anterior se ha marcado algún símbolo, se repite de nuevo dicho paso teniendo en cuenta los símbolos marcados. En caso contrario, fin del algoritmo.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

97

Gramáticas Bien Formadas

1. **Limpieza de Gramáticas**
   b. **Símbolos Inaccesibles**

Ejemplo

sea la G = ({a, b, c}, {S,A,B,C}, S, P),

donde P = {S::= aA
A ::= Bc
B::= bA
C::= c}

1. {a, b, c, S, A, B, C}
2. {a, b, c, S, A, B, C}
3. {a, b, c, S, A, B, C}
4. {a, b, c, S, A, B, C}
5. {a, b, c, S, A, B, C}
6. {a, b, c, S, A, B, C}

Sin cambios.
Fin algoritmo

C es un símbolo INACCESIBLE!!

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

98

Gramáticas Bien Formadas

1. **Limpieza de Gramáticas**

   c. **Reglas Superfluas**: son aquellas que no contribuyen a la formación de palabras x $\in \Sigma_T^*$.

   Estas reglas contiene algún **símbolo No Terminal no generativo**.

   Todo símbolo no superfluo debe cumplir U $+\!\!\rightarrow$t, tal que t $\in \Sigma_T^*$

Algoritmo (es un algoritmo recursivo de marcado)

a. marcar los NT para los que existe una regla U ::= x donde x $\in \Sigma^*$ (es una cadena de T o λ, o en pasadas sucesivas contiene NT marcados)
b. si todos los NT están marcados $\Rightarrow$ no existen símbolos superfluos y fin
c. si la última vez que se pasó por el paso a) se marcó un NT, volver al paso a).
d. todo A $\in \Sigma_{NT}$ no marcado es superfluo.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

99

Gramáticas Bien Formadas

1. **Limpieza de Gramáticas**

   c. **Reglas Supérfluas**:

   ejemplo: sea G= ({e,f}, {S,A,B,C,D}, S, P)

   donde P= {

   S::= Be

   A::= Ae |e

   B::= Ce | Af

   C::= Cf

   D::= f}

1ª pasada:

D::= f y A::= e

Es una cadena de Terminales

2ª pasada:

B::= Af

Los NoTerminales están marcados.

3ª pasada:

S::= Be

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

100

Gramáticas Bien Formadas

1. **Limpieza de Gramáticas**
   c. **Reglas Supérfluas**:

ejemplo: sea G= ({e,f}, {S,A,B,~~C~~,D}, S, P)

donde P= {

S::= Be

A::= Ae |e

B::= ~~Ce~~ | Af

C:~~:=~~Cf

D::= f}

1ª pasada:

D::= f y A::= e

2ª pasada:

B::= Af

3ª pasada:

S::= Be

Sin marcar: C, que se elimina, así como sus reglas

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

101

Gramáticas Bien Formadas

Transformación de una G dada en otra equivalente cuyas reglas de producción estén en un formato carente de imperfecciones:

1. **Limpieza de Gramáticas**
   a. Reglas Innecesarias
   b. Símbolos Inaccesibles
   c. Reglas Superfluas
2. **Eliminación de símbolos no generativos**
3. **Eliminación de reglas no generativas**
4. **Eliminación de reglas de redenominación**

Gramática Reducida

Gramática Limpia

Gramática Bien Formada

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

102

Gramáticas Bien Formadas

2. **Eliminación de símbolos no generativos:**

   Sea G2 = (Σ_T, Σ_N, S, P), $\forall A \in \Sigma_N$ construiremos la gramática G(A), donde A es el axioma. Si **L(G(A))= $\phi \Rightarrow$ A es símbolo no generativo** y se puede eliminar, así como todas las reglas que lo contengan, obteniéndose otra G2 equivalente.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

3. **Eliminación de reglas no generativas:** son A::= λ (A$\neq$S)

   Algoritmo:

   1. Eliminar de la Gramática las regla de la forma U::= λ
   2. Por cada regla de la gramática donde U aparezca en la parte dcha., V::=xUy, se añade la regla V::=xy (a menos que ya existe).
   3. Repetir 2. hasta que no quede ninguna regla de la forma U::= λ, o que sólo quede S::= λ.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

3. Eliminación de reglas no generativas: son A::= λ (A≠S)

Ejemplo:

G = ({a, b}, {S, P, Q}, S, Pr)

Pr = { S ::= PQ | aSb | P

P ::= aPQ | a

Q ::= Qb | λ }

Regla NO generativa:
Q ::= λ

ELIMINARLA

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

3. Eliminación de reglas no generativas: son A::= λ (A≠S)

Ejemplo:

G = ({a, b}, {S, P, Q}, S, Pr)

Pr = { S ::= PQ | aSb | P

P ::= aPQ | a

Q ::= Qb | λ }

Eliminar la Regla No generativa

G' = ({a, b}, {S, P, Q}, S, Pr)

Pr = { S ::= PQ | aSb | P

P ::= aPQ | a | aP

Q ::= Qb | b }

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

Transformación de una G dada en otra equivalente cuyas reglas de producción estén en un formato carente de imperfecciones:

1. **Limpieza de Gramáticas**
   a. Reglas Innecesarias
   b. Símbolos Inaccesibles
   c. Reglas Supérfluas
2. **Eliminación de símbolos no generativos**
3. **Eliminación de reglas no generativas**
4. **Eliminación de reglas de redenominación**

Gramática Reducida

Gramática Limpia

Gramática Bien Formada

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

4. Eliminación de reglas de redenominación: son reglas del tipo A::=B (donde A ≠ B)

Algoritmo

1. Eliminar de la Gramática las regla de la forma U::= V
2. Por cada regla de la forma V::=x, añadir U::=x (si No Existe).
3. Repetir 2. hasta que no quede ninguna regla de redenominación.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

4. **Eliminación de reglas de redenominación:** son reglas del tipo A::=B (donde A ≠ B)

Ejemplo:

G = ({a, b}, {S, P, Q}, S, Pr)

Pr = { S ::= PQ | aSb | P

P ::= aPQ | aP | a

Q ::= Qb | b }

Regla de Redenominación: S ::= P

ELIMINARLA

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

4. **Eliminación de reglas de redenominación:** son reglas del tipo A::=B (donde A ≠ B)

Ejemplo:

G = ({a, b}, {S, P, Q}, S, Pr)

Pr = { S ::= PQ | aSb | P

P ::= aPQ | aP | a

Q ::= Qb | b }

Eliminar la Redenominación

G' = ({a, b}, {S, P, Q}, S, Pr)

Pr = { S ::= PQ | aSb | aPQ | aP | a

P ::= aPQ | aP | a

Q ::= Qb | b }

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Gramáticas Bien Formadas

Transformación de una G dada en otra equivalente cuyas reglas de producción estén en un formato carente de imperfecciones:

1. **Limpieza de Gramáticas**
   a. Reglas Innecesarias
   b. Símbolos Inaccesibles
   c. Reglas Supérfluas
2. **Eliminación de símbolos no generativos**
3. **Eliminación de reglas no generativas**
4. **Eliminación de reglas de redenominación**

Gramática Reducida

Gramática Limpia

Gramática Bien Formada

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

- Gramáticas Tipo 2
- Gramáticas bien formadas
- Formas normales

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Formas Normales

✓ Son notaciones que se aplican a las G2:

Afectan a la forma de las reglas de producción

✓ Son dos las que se va a estudiar:

Forma Normal de Chomsky

Forma Normal de Greibach

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Forma Normal de Chomsky

Una gramática bien formada está en Forma Normal de Chomsky (FNC) si las partes derechas de todas sus reglas de producción tienen a lo sumo dos símbolos, y cuando son dos, ambos son No Terminales.

Ejemplo: La siguiente gramática está en FNC:

S ::= λ | a | NN

N ::= NT | AB

A ::a

B::= b

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

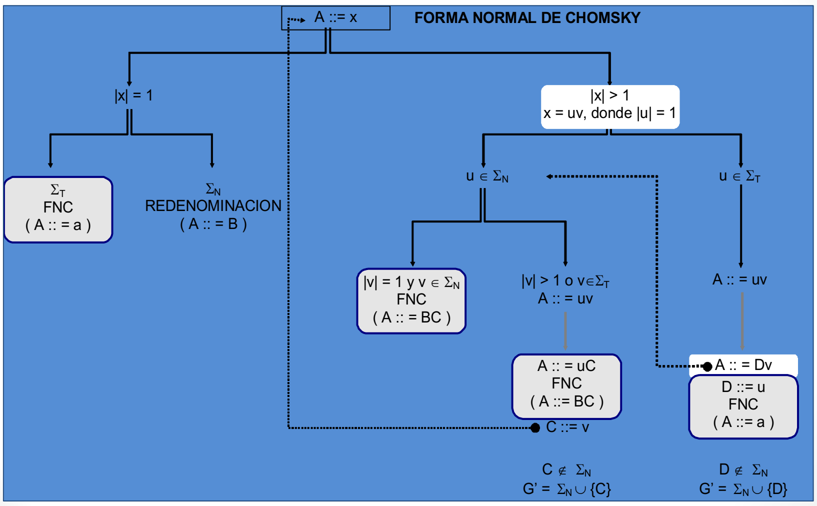
Forma Normal de Chomsky

A partir de cualquier gramática de Tipo 2, se puede construir otra equivalente que está en FNC.

Para ello, se deben sustituir del siguiente modo las reglas cuya parte derecha tiene más de 2 símbolos por varias reglas que tengan en la parte derecha dos símbolos no terminales o un solo símbolo terminal:

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Forma Normal de Chomsky



A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Forma Normal de Chomsky

EJEMPLO:

G = ({a, b, c}, {A, B, C, D, E, F}, A, P)
P =
B::=B
C::=C
A::=ABC
D::=a
E::=FbB
F::=EaA
A::=bA
B::=Ba
A::=aBb
B::=bAa
A::=a
B::=a
C::=c
C::=Cb

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

117

Forma Normal de Chomsky

Bien Formar la Gramática

G = ({a, b, c}, {A, B, C, D, E, F}, A, P)
P =
B::=B ←Innecesaria
C::=C ←Innecesaria
A::=ABC
D::=a ←Innaccesible
E::=FbB ←Innaccesible
F::=EaA ←Innaccesible
A::=bA
B::=Ba
A::=aBb
B::=bAa
A::=a
B::=a
C::=c
C::=Cb

Gramática Bien Formada:
G = ({a, b, c}, {A, B, C}, A, P)
P =
A::=a
A::=aBb
A::=ABC
A::=bA
B::=a
B::=Ba
B::=bAa
C::=c
C::=Cb

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

118

Forma Normal de Chomsky

Gramática Bien Formada:
G = ({a, b, c}, {A, B, C, D, E, F}, A, P)
P =

A::=a	Sí (FNC)
A::=aBb	No (FNC)
A::=ABC	No
A::=bA	No
B::=a	Sí
B::=Ba	No
B::=bAa	No
C::=c	Sí
C::=Cb	No

Transformación a FNC (quedan tabuladas las nuevas reglas generadas en FNC)

Tratamiento de A::=aBb
A::=aBb
  D::=a
A::=DBb
E::=Bb
  A::=DE
  F::=b
  E::=BF

Tratamiento de A::=ABC
A::=ABC
  G::=BC
  A::=AG

Tratamiento de A::=bA
A::=bA
  A::=FA

Tratamiento de B::=Ba
B::=Ba
  B::=BD

Tratamiento de B::=bAa
B::=bAa
B::=FAa
H::=Aa
B::=FH
H::=AD

Tratamiento de C::=Cb
C::=Cb
  C::=CF

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

119

Forma Normal de Greibach

✓ **FNG** es una notación muy interesante para algunos reconocimientos sintácticos. En ella todas las reglas tienen la parte derecha comenzando con un terminal seguido opcionalmente de uno o varios NT

1. TEOREMA: todo **L de contexto libre sin** λ puede ser generado por una G2 en la que todas las reglas sean de la forma:
   - **A → a**α donde A $\in \Sigma_{NT}$,a$\in \Sigma_T$ y $\alpha \in \Sigma_{NT}^*$
   - Si $\lambda \in$ L habrá que añadir **S::=** λ
2. TEOREMA: toda G2 puede reducirse a otra G2 equivalente sin reglas recursivas a izquierdas

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

120

Forma Normal de Greibach

FNG: para transformar una G2 en su equivalente en forma normal de Greibach:

1. Limpiar y formar bien. Eliminar la recursividad a izquierdas
2. Aplicar el algoritmo de transformación a FNG, verificando en cada paso que no aparezcan nuevas reglas recursivas a izquierdas y si aparecen, eliminándolas con el paso 1

EJEMPLO:
G = ({a,b}, {S}, S, P), donde P = {S::=aSb | SS | λ}

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Forma Normal de Greibach

1. Eliminar la recursividad a izquierdas, resumiendo, sería:

   sea G = ({α_1, α_2, β_1, β_2}, {A}, A, P),

   donde P = {A::= A α_1| Aα_2 | β_1| β_2}

   Quedaría:

   A::= β_1 | β_2 | β_1X | β_2X
   X::= α_1| α_2 |α_1X| α_2X

Eliminar la recursividad a izquierdas:
S ::= aSb | SS | λ → (se transforma en) → S ::= aSb | aSbX | λ y X ::= SX | S

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Universidad Carlos III de Madrid

Forma Normal de Greibach

2.Transformación de G2 bien formada sin RI a FNG:

2.1 Establecer una relación de orden parcial en Σ_{NT}

$\Sigma_{NT} = \{A_1, A_2, ..., A_n\}$ basándose en: si Ai $\rightarrow$ Aj α, Ai precederá a Aj.

Cuando hay reglas "contradictorias" usar una de ellas para el orden y mirar el resto para ver que conviene más

A ::= αBβ (A sería nº 1 y B nº 2). α es una cadena de 0 o más terminales y β una cadena de 0 o más símbolos.

B ::= δCγ (B sería nº 2 y C nº 3). δ es una cadena de 0 o más terminales y γ una cadena de 0 o más símbolos.

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Forma Normal de Greibach

2.Transformación de G2 bien formada sin RI a FNG:

2.2 Se clasifican las reglas en 3 grupos:

Grupo 1: Ai $\rightarrow$ a α ,donde $a \in \Sigma_T$ y $\alpha \in \Sigma^*$

Grupo 2: Ai $\rightarrow$ Aj α donde Ai precede a Aj en el conjunto Σ_{NT} ordenado

Grupo 3: Ak $\rightarrow$ Ai α donde Ai precede a Ak en el conjunto Σ_{NT} ordenado

Hacer lo mismo con las de grupo 2

Ordenar los no terminales: {X, S} y clasificar las reglas según este orden:
S ::= aSb (G1, aunque hay que quitar el terminal que está detrás del no terminal)
S ::= aSbX (G1, aunque hay que quitar el terminal que está detrás del no terminal)
S ::= λ (G1)
X ::= SX (G2)
X ::= S (G2)

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Universidad Carlos III de Madrid

Forma Normal de Greibach

2.Transformación de G2 bien formada sin RI a FNG:

2.3 Se transforman las reglas de grupo 3 → grupo 2 → grupo 1: FNG

Ak → Ai α se sustituye **Ai** por la parte dcha. de todas las reglas que tienen **Ai** como parte izda.

Hacer lo mismo con las de grupo 2

Ordenar los no terminales: {X, S} y clasificar las reglas según este orden:
S ::= aSb (G1, aunque hay que quitar el terminal que está detrás del no terminal)
S ::= aSbX (G1, aunque hay que quitar el terminal que está detrás del no terminal)
S ::= λ (G1)
X ::= SX (G2)
X ::= S (G2)

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Forma Normal de Greibach

2.Transformación de G2 bien formada sin RI a FNG:

2.3 Se transforman las reglas de grupo 3 → grupo 2 → grupo 1: FNG

Ak → Ai α se sustituye **Ai** por la parte dcha. de todas las reglas que tienen **Ai** como parte izda.

Es decir, Sustituir el primer símbolo NT de la parte dcha. de cada regla del grupo 3 por las partes dchas. de todas las reglas (en cualquier grupo) donde dicho primer símbolo aparezca como parte izquierda. Hacer esto hasta que hayamos conseguido que todas las reglas del grupo 3 hayan sido transformadas en reglas de otros grupos. Si en este proceso aparecen reglas recursivas a izquierdas, transformarlas.

Hacer lo mismo con las de grupo 2

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Forma Normal de Greibach

2.Transformación de G2 bien formada sin RI a FNG:

2.3 Se transforman las reglas de grupo 3 → grupo 2 → grupo 1: FNG

Ak → Ai α se sustituye **Ai** por la parte dcha. de todas las reglas que tienen **Ai** como parte izda.

Hacer lo mismo con las de grupo 2

Transformamos las de grupo G2 en grupo G1:

X ::= SX **→S ::= aSb→ X ::= aSbX (G1, aunque hay que quitar el terminal que está detrás del no terminal)**
→S ::= aSbX→ X ::= aSbXX (G1, aunque hay que quitar el terminal que está detrás del no terminal)

X ::= S **→S ::= aSb→ X ::= aSb (G1, aunque hay que quitar el terminal que está detrás del no terminal)**
→S ::= aSbX→ X ::= aSbX (G1, aunque hay que quitar el terminal que está detrás del no terminal)

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Forma Normal de Greibach

2. Transformación de G2 bien formada sin RI a FNG:

2. 4. Cuando todas las reglas son de grupo 1, la G está en FNG a falta de **eliminar los símbolos terminales no situados en la cabecera de la parte derecha.**

$\boxed{A \rightarrow \alpha\ a\ \beta}$ **donde** $a \in \Sigma_T$ **y** $\alpha \neq \lambda$

↓

$$\boxed{\begin{array}{l} A \rightarrow \alpha\ B\ \beta \\ B \rightarrow a \end{array}}$$

Es decir, transformar los símbolos terminales no situados en cabeza de parte derecha, asignándoles nuevos NT que deriven en ellos (o NT que ya existen que sólo deriven en ellos).

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Forma Normal de Greibach

2. Transformación de G2 bien formada sin RI a FNG:

2. 4. Cuando todas las reglas son de grupo 1, la G está en FNG a falta de **eliminar los símbolos terminales no situados en la cabecera de la parte derecha.**

$\boxed{A \rightarrow \alpha\ a\,\beta}$ **donde** $a \in \Sigma_T$ **y** $\alpha \neq \lambda$

$$\boxed{\begin{array}{l} A \rightarrow \alpha\ B\,\beta \\ B \rightarrow a \end{array}}$$

Introducimos un nuevo símbolo para quitar la a:

B ::= a

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Forma Normal de Greibach

EJEMPLO

G = ({a,b}, {S}, S, P),

P = {S::=aSb | SS | λ}

La Gramática en FNG queda:

G' = ({a,b}, {S,X,B}S, P'})

P' = {
S ::= aSB | aSBX | λ
X ::= aSBX | aSBXX | aSB
B ::= b
}

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Bibliografía

- Libro Básico 1 Bibliografía (AAM). Enrique Alfonseca Cubero, Manuel Alfonseca Cubero, Roberto Moriyón Salomón. Teoría de autómatas y lenguajes formales. McGraw-Hill (2007). Capítulo 5

- Libro Básico 2 Bibliografía (HMU). John E. Hopcroft, Rajeev Motwani, Jeffrey D.Ullman. Introducción a la teoría de autómatas, lenguajes y computación (3ª edición). Ed, Pearson Addison Wesley.

- Libro Básico 4 Bibliografía (AAM). Manuel Alfonseca, Justo Sancho, Miguel Martínez Orga. Teoría de lenguajes, gramáticas y autómatas. Publicaciones R.A.E.C. 1997 Capítulo 3

A. Sanchis, A. Ledezma, J. Iglesias, B. García, J. Alonso

Parte V

TEMA 5. Expresiones Regulares

5. Lenguajes Regulares

Grado Ingeniería Informática
Teoría de Autómatas y Lenguajes Formales

AUTÓMATAS FINITOS Y G3

2

Gramática asociada a un AF

Sea el AF, A = (Σ, Q, q0, f, F), existe una G3 LD tal que

L(G3LD) = L(A)

Es decir, el lenguaje que genera la gramática es el mismo que reconoce el Autómata

Veamos como se obtiene la gramática G={ΣT, ΣN, S, P} a partir del AF= {Q, Σ, q0, f, F}.

3

Gramática asociada a un AF

Se construye la **gramática G3LD** (G={Σ_T, Σ_N, S, P}) de la siguiente forma, **a partir del Autómata** (AF= {Σ, Q, q_0, f, F}):

- $\Sigma_T = \Sigma$; $\Sigma_N = Q$; $S = q_o$
- P= { … }
  1. transición f(p,a) = q → si q ∉ F→ **p::= a q**
  2. Si q ∈ F y f(p,a) = q → **p::= a** y **p::= a q**
  3. Si q_0 ∈ F → **q_0 ::= λ (es axioma para dar** λ)
  4. si f(p, λ) = q → si q ∉ F → p ::= q (redenominación);
  5. q ∈ F y f(p, λ) = q → p ::= q y p::= λ (redenominación y no generativa)

4

AF asociado a una G3 (cuando es LD)

De AF → G3: Ejemplo

Sea el AF descrito por la siguiente tabla, hallar la G3LD que genera el lenguaje por ella descrito. Comprobar que los lenguajes son iguales

	0	1
→A	A	C
B	A	C
C*	C	B

AF asociado a una G3LD

Sea la G3LD, G=(Σ_T, Σ_N S, P), existe un AF, A, tal que:

L(G3LD) = L(A)

Cómo se construye **AF a partir de G3LD**:

- $\Sigma = \Sigma_T$
- $Q = \Sigma_N \cup \{F\}$, con $F \notin \Sigma_N$
- $q_0 = S$
- F = {F}
- f:

Si A ::= a B → f(A,a) = B
Si A ::= a → f(A,a) = F
Si S ::= λ → f(S, λ) = F (equivalente a hacer $q_0 = S \in F$)

AF asociado a una G3 (cuando es LD)

Se ha visto el procedimiento para obtener el AF que aceptaba el lenguaje descrito por una G3LD, sin embargo, ese procedimiento no siempre conduce a un AFD.

Lo habitual es: G3LD → AFND → AFD

1. Ejemplo: Sea la G3LD hallar el AF correspondiente.

G = ({d,c}, {A,S,T}, A, {A ::= cS, S::= d|cS|dT, T::= dT|d})

7

AF asociado a una G3

¿Y si queremos obtener un AF a partir de una G3LI?

G3LI → G3LD → AF

¿Y si queremos obtener una G3LI a partir de un AF?

AF → G3LD → G3LI

8

EXPRESIONES REGULARES

- Definición
- Expresiones y Lenguajes Regulares
- Equivalencias
- Teoremas

Definición de ER (I)

“ Metalenguaje para expresar el conjunto de palabras aceptadas por un AF (es decir, para expresar lenguajes de tipo 3 o regulares)”

Kleene, 1956

11

Definición de ER(I)

Ejemplo

Dado el alfabeto Σ= {0,1},

La ER 0*10* es una palabra del metalenguaje que representa las infinitas palabras del lenguaje regular formado por un 1, precedido y seguido de 0, 1 o infinitos 0s.

El lenguaje Σ* puede representarse mediante la ER:

(0+1)*

El lenguaje {01, 101} puede representarse mediante la ER:

01 + 101

La ER 1(1+0)* representa todas las cadenas que empiezan por el símbolo 1.

12

Definición de ER(II)

Dados los elementos/símbolos:

Σ, $\varnothing$ (lenguaje vacío), λ (palabra vacía)

y las operaciones:

$+$ (unión), •(concatenación), *(cierre o clausura)

se cumple que:

- $\varnothing$ es una ER
- λ es una ER
- cualquier $a \in \Sigma$ es una ER
- si α y β son EERR entonces $\alpha+\beta$ y $\alpha\bullet\beta$ son EERR
- si α es una ER entonces α^* es una ER, donde $\alpha^* = \bigcup_{i=0}^{\infty} \alpha^i$

13

Definición de ER(III)

Solo son EERR las que se obtienen de aplicar las reglas anteriores **un número finito de veces** sobre símbolos de Σ, $\varnothing$, λ

La prioridad de las operaciones es la siguiente:

$* > \bullet > +$

14

- Definición
- Expresiones y Lenguajes Regulares
- Equivalencias
- Teoremas

15

EERR y LR

Cada Expresión Regular (ER) describe o expresa un lenguaje regular

A cada ER α, se le asocia un subconjunto de Σ^*, $L(\alpha)$, que es el LR descrito por α. Este lenguaje se define con:

- si $\alpha = \varnothing$, $L(\alpha) = \varnothing$
- si $\alpha = \lambda$, $L(\alpha) = \{\lambda\}$
- si $\alpha = a$, $a \in \Sigma$, $L(\alpha) = \{a\}$
- si α y β son EERR $\Rightarrow L(\alpha + \beta) = L(\alpha) \cup L(\beta)$
- si α y β son EERR $\Rightarrow L(\alpha \bullet \beta) = L(\alpha)\, L(\beta)$
- si α * es una ER $\Rightarrow L(\alpha^*) = L(\alpha)^*$

16

- Definición
- Expresiones y Lenguajes Regulares
- Equivalencias
- Teoremas

Equivalencia de EERR (I)

Dos EERR son equivalentes, $\alpha = \beta$, si describen el mismo lenguaje regular, si $L(\alpha) = L(\beta)$

Se cumple:

1) $(\alpha + \beta) + \sigma = \alpha + (\beta + \sigma)$ (+ es asociativa)
2) $\alpha + \beta = \beta + \alpha$ (+ es conmutativa)
3) $(\alpha \bullet \beta) \bullet \sigma = \alpha \bullet (\beta \bullet \sigma)$ (• es asociativa)
4) $\alpha \bullet (\beta + \sigma) = (\alpha \bullet \beta) + (\alpha \bullet \sigma)$ (+ es distributiva
   $(\beta + \sigma) \bullet \alpha = (\beta \bullet \alpha) + (\sigma \bullet \alpha)$ respecto de •)
5) $\alpha \bullet \lambda = \lambda \bullet \alpha = \alpha$ (• tiene elemento neutro)
6) $\alpha + \varnothing = \varnothing + \alpha = \alpha$ (+ tiene elemento neutro)
7) $\lambda^* = \lambda$
8) $\alpha \bullet \varnothing = \varnothing \bullet \alpha = \varnothing$

Equivalencia de EERR (II)

9) $\varnothing^* = \lambda$

10) $\alpha^* \bullet \alpha^* = \alpha^*$

11) $\alpha \bullet \alpha^* = \alpha^* \bullet \alpha$

12) $(\alpha^*)^* = \alpha^*$ (IMPORTANTE)

13) $\alpha^* = \lambda + \alpha + \alpha^2 + .. + \alpha^n + \alpha^{n+1}. \alpha^*$

14) $\alpha^* = \lambda + \alpha \bullet \alpha^*$ (13 con n=0) (IMPORTANTE)

15) $\alpha^* = (\lambda + \alpha)n\text{-}1 + \alpha n \bullet \alpha^*$ (de 14, sustituyendo)

16) Sea f una función, $f{:}E^n{}_\Sigma \rightarrow E_\Sigma$ se verifica:

$$f(\alpha, \beta, \ldots, \sigma) + (\alpha + \beta + \ldots + \sigma)^* = (\alpha + \beta + \ldots + \sigma)^*$$

17) Sea f una función, $f{:}E^n{}_\Sigma \rightarrow E_\Sigma$ se verifica:

$$(f(\alpha^*, \beta^*, \ldots, \sigma^*))^* = (\alpha + \beta + \ldots + \sigma)^*$$

19

Equivalencia de EERR (III)

18) $(\alpha^* + \beta^*)^* = (\alpha^* \bullet \beta^*)^* = (\alpha + \beta)^*$ (IMPORTANTE)

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

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

21) $(\alpha^* \bullet \beta)^* = \lambda + (\alpha + \beta)^* \bullet \beta$ (de 14 con 20)

22) Reglas de Inferencia:

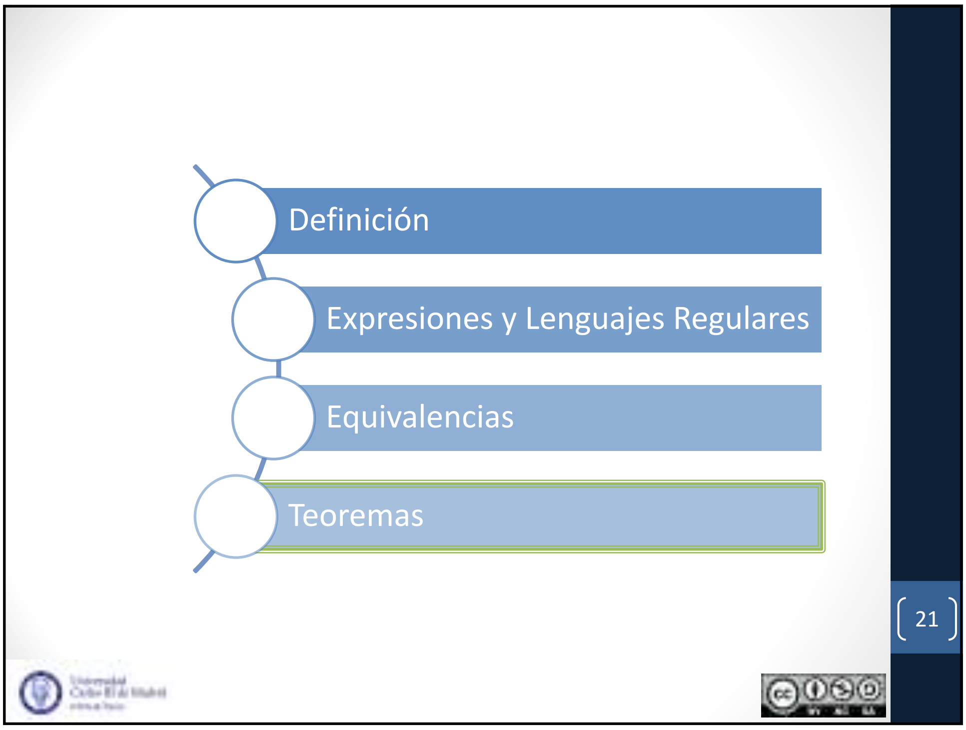
Dadas tres EERR (L, A y B), sea la ecuación

L = AL + B,

donde $\lambda \notin A$, entonces se verifica que

L = A*B

20



Teoremas de análisis y síntesis de Kleene

1. **Teorema de análisis de Kleene**

Todo lenguaje aceptado por un AF es un lenguaje regular.

Solución al problema de análisis:

Encontrar el lenguaje asociado a un determinado AF: "Dado un AF, A, encontrar la ER que describe L(A)".

1. **Teorema de síntesis de Kleene**

Todo lenguaje regular es el lenguaje aceptado por un AF.

Solución al problema de síntesis:

Encontrar un reconocedor para un lenguaje regular dado: *"Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular"*.

22

1. Solución al problema de análisis. Ecuaciones características.

Problema Análisis: AF -> ER

Resolución:

Dado un AF, escribir las ecuaciones características de cada uno de sus estados, resolverlas y obtener la ER buscada.

23

Solución al problema de análisis. Ecuaciones características.

ECUACIONES CARACTERÍSTICAS:
Describen todas las cadenas que se pueden reconocer desde un estado dado

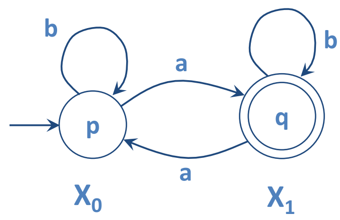
Se escribe una ecuación x_i por estado q_i

- Primer miembro: $\mathbf{x_i}$
- El segundo miembro tiene un término por cada rama que salga de q_i
  - Las ramas tienen la forma $\mathbf{a_{ij} \bullet x_j}$ donde a_{ij} es la etiqueta de la rama que une q_i con q_j, x_j es la variable correspondiente a q_j
  - **Se añade un término a_{ij} por cada rama que une qi con un estado final**
  - ~~**Se añade λ si q_i es final.**~~ **SOLO si es final sin ramas o SOLO ramas al sumidero**
  - Si de un estado q_i no sale ninguna rama, el segundo miembro será:
    - si es final: $x_i = \lambda$
    - si no es final: $x_i = \varnothing$

24

Solución al problema de análisis. Ecuaciones características.

Ecuaciones Características de un AF – Ejemplo 1:



El AF tiene 2 estados, por lo que tendrá 2 ecuaciones características:

Conjunto de palabras que permiten pasar desde el estado *p* a un estado final.

$X_0 = b\,X_0 + a\,X_1 + a$

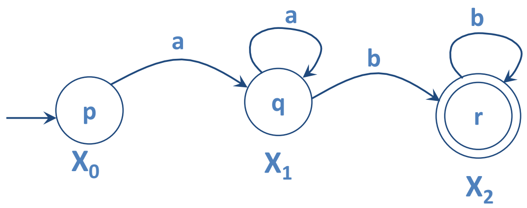
Porque *q* es un estado final

$X_1 = b\,X_1 + a\,X_0 + b$

Porque *q* es un estado final

Solución al problema de análisis. Ecuaciones características.

Ejemplo 2:



Ecuaciones Características:

$X_0 = a\,X_1$

$X_1 = b\,X_2 + a\,X_1 + b$

$X_2 = b\,X_2 + b$

Algoritmo de resolución problema de Análisis.

1. Escribir las ecuaciones características del AF
2. Resolverlas
3. Si el estado inicial es q_0, X_0 nos da el conjunto de cadenas que conducen desde q_0 a q_f y por tanto el lenguaje aceptado por el AF

27

Solución de las ecuaciones características.

La Ecuación Característica de la forma: X = AX + B, donde:

X: conjunto de cadenas que permiten pasar de q_i a $q_f \in F$

A: conjunto de cadenas que permiten, partiendo de un estado q, llegar a q.

B: conjunto de cadenas que permiten llegar al estado final, sin volver a pasar por el q_i de partida.

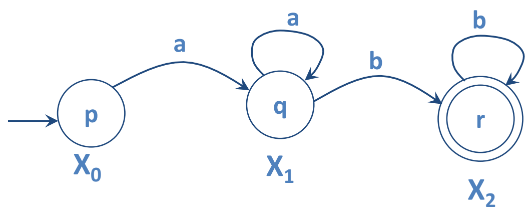
$\Downarrow$ (solución de Arden o reducción al absurdo)

La solución es: **X = A* • B**

28

Solución al problema de análisis. Ecuaciones características.

Ejemplo 1:



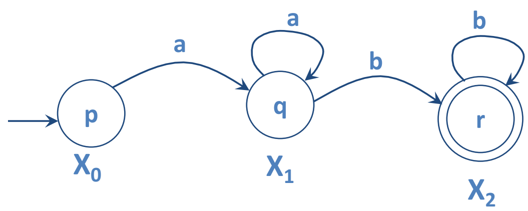
Ecuaciones Características:

$X_0 = a\, X_1$
$X_1 = b\, X_2 + a\, X_1 + b$
$X_2 = b\, X_2 + b$

29

Solución al problema de análisis. Ecuaciones características.

Ejemplo 1:



Recuerda:
L=AL+B

L=A*B

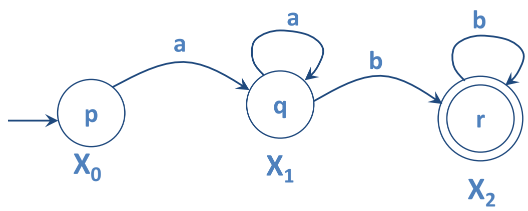
Ecuaciones Características:

$X_0 = a\, X_1$
$X_1 = b\, X_2 + a\, X_1 + b$
$X_2 = b\, X_2 + b + \lambda$

30

Solución al problema de análisis. Ecuaciones características.

Ejemplo 1:



Recuerda:
L=AL+B

L=A*B

Ecuaciones Características:

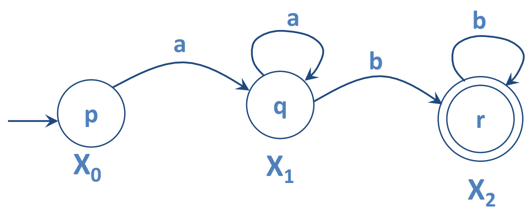
$X_0 = a\,X_1$
$X_1 = b\,X_2 + a\,X_1 + b$
$X_2 = b\,X_2 + b$

$X_2 = b^*\,b$

31

Solución al problema de análisis. Ecuaciones características.

Ejemplo 1:



Recuerda:
L=AL+B

L=A*B

Ecuaciones Características:

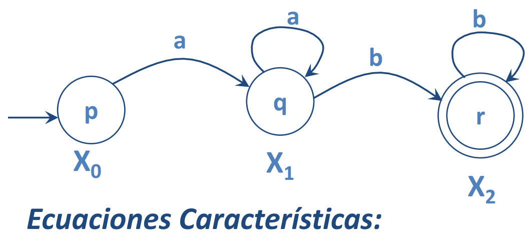
$X_0 = a\,X_1$
$X_1 = b\,X_2 + a\,X_1 + b$
$X_2 = b\,X_2 + b$

$X_2 = b^*\,b$
$X_1 = bb^*\,b + aX_1 + b$
$X_1 = aX_1 + bb^*\,b + b$
$X_1 = a^*(b\,b^*\,b + b) =$
$a^*\,(b(b^*b+\lambda\,) = a^*bb^*$

32

Solución al problema de análisis. Ecuaciones características.

Ejemplo 1:



Recuerda:
L=AL+B

L=A*B

Ecuaciones Características:

$X_0 = a\, X_1$
$X_1 = b\, X_2 + a\, X_1 + b$
$X_2 = b\, X_2 + b$

$X_1 = a^*bb^*$
$X_0 = a\,(X_1) = aa^*bb^*$

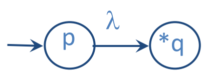
33

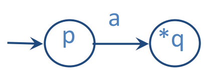
2. Problema de Síntesis: Algoritmo Recursivo (I)

Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular.

Sea α una Expresión Regular

- si $\alpha = \varnothing$, el autómata será: 

- si $\alpha = \lambda$, el autómata será: 

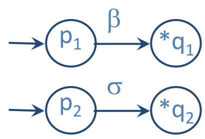
- si $\alpha = a$, $a \in \Sigma$, el autómata será: 

34

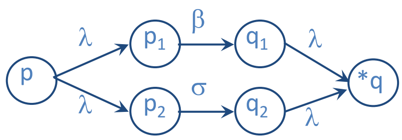
Problema de Síntesis: Algoritmo Recursivo (II)

Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular *(cont.)*

- si $\alpha=\beta+\sigma$, con los autómatas de β y σ



el resultado es:

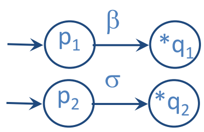


35

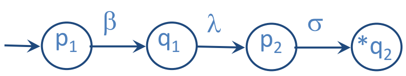
Problema de Síntesis: Algoritmo Recursivo (III)

Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular. (*cont.)*

- si $\alpha=\beta \bullet \sigma$, con los autómatas de β y σ



el resultado es:

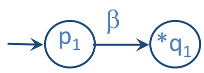


36

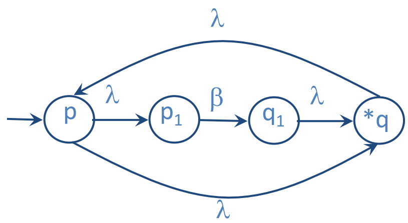
Problema de Síntesis: Algoritmo Recursivo (IV)

Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular. *(cont.)*

- si $\alpha=\beta^*$, con el autómata de β



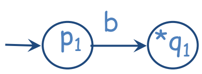
el resultado es:



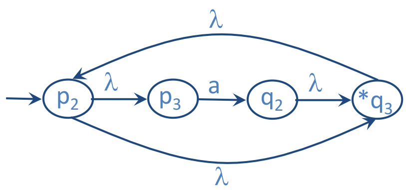
Problema de Síntesis: Algoritmo Recursivo (IV)

Ejemplo: Sea α= b • a*

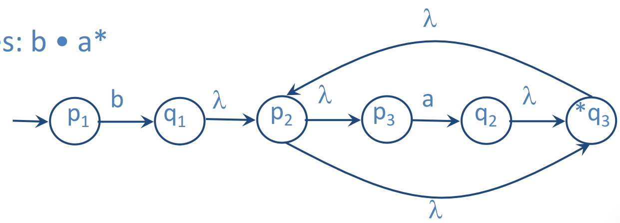
b:



a*:



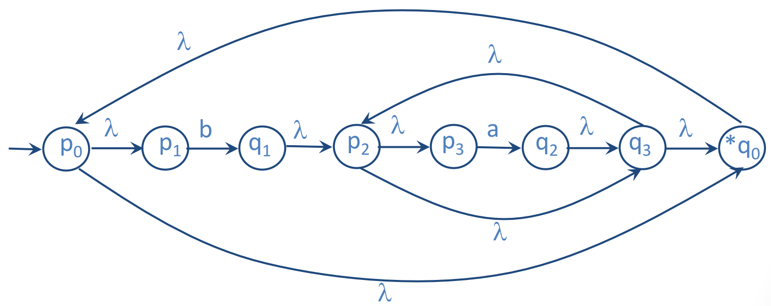
Entonces: b • a*



Problema de Síntesis: Algoritmo Recursivo (IV)

Ejemplo:

Sea $\alpha=(b \bullet a^*)^*$



39

2. Problema de Síntesis: Derivada de una ER.

Dada una ER, construir un AF que reconozca el lenguaje que la ER describe.

Solución: derivar la ER y obtener una G3LD y de ella un AF

Derivada de una ER: $Da(R) = \{ x \mid a \bullet x \in R \}$.

- Derivada de ER R respecto de $a \in \Sigma$ es el conjunto de colas de todas las palabras representadas por R cuya cabeza es a.

Veamos una definición recursiva

40

Problema de Síntesis: Derivada de una ER

ER -> AF (Derivar la ER -> G3LD -> AF. $Da(R) = \{ x \mid a \bullet x \in R \}$

Derivada de una ER. ***Definición recursiva***

$\forall$ a, b $\in \Sigma$ y R,S expresiones regulares

- $Da(\varnothing) = \varnothing$
- $Da(\lambda) = \varnothing$
- $Da(a) = \lambda, \quad a \in \Sigma$
- $Da(b) = \varnothing, \quad \forall\, b \neq a,\, b \in \Sigma$
- $Da(R+S) = Da(R) + Da(S)$
- $Da(R \bullet S) = Da(R) \bullet S + \delta(R) \bullet Da(S) \quad \forall R$
  - $\lambda \in R \Rightarrow \delta(R) = \lambda$
  - $\lambda \notin R \Rightarrow \delta(R) = \varnothing$
- $Da(R^*) = Da(R) \bullet R^*$

41

Solución al problema de Síntesis. Derivada de una ER

- **Definición: Dab(R) = Db(Da(R))**
- A partir de la Derivada de una ER. Se obtendrá la gramática regular lineal derecha:
  - El número de derivadas distintas de una ER es finito.
    Una vez que se han obtenido todas, se puede obtener la G3
  - **Si δ (R0) = λ añadimos R0 → λ**
  - Sea Da(R) = S, con $S \neq \Phi$
    - $S \neq \lambda \Rightarrow R ::= aS \in P$
    - $S = \lambda \Rightarrow R ::= a \in P$
  - Sea $\delta(Da(R)) = S$
    - $\delta(Da(R)) = \lambda \Rightarrow R ::= a \in P$
    - $\delta(Da(R)) = \Phi \Rightarrow$ no se incluye ninguna regla en P
  - El axioma es R (ER de partida)
    - Σ_T = símbolos que formaban la ER de partida
    - Σ_N = letras que distinguen cada una de las derivadas distintas

42

Ejemplos. Derivada Expresiones Regulares

Obtener las G3 LD equivalentes a las ER dadas:

R = a a* b b*, Σ={a,b}

R = a a* b b* es igual que
R = a • a* • b • b*

- Da(R) = Da(a) a* b b* = a* b b*
- Db(R) = ∅
- Daa(R) = Da(a* b b*) = Da(a*) b b* + λ Da(b b*) = a*bb* = Da(R)
- Dab(R) = Db(a* b b*) = Db(a*) b b* + λ Db(b b*) = b*
- Daba(R) = Da(b*) = ∅
- Dabb(R) = Db(b*) = Db(b) b* = b* = Dab(R)

- Da(R)= a*bb* δ(Da(R))= ∅
- Daa(R)= a*bb* δ(Daa(R))= ∅
- Dab(R)= b* δ(Dab(R))= λ
- Dabb(R)= b* δ(Dabb(R))= λ

43

Ejemplos. Derivada Expresiones Regulares

- Obtención de la G3 LD que genera el Lenguaje representado por la ER:
- R0=aa*bb* R1=a*bb* R2=b*
  - Da(R0)=R1 δ(Da(R0))= ∅
  - Da(R1)=R1 δ(Da(R1))= ∅
  - Db(R1)=R2 δ(Db(R1))= λ
  - Db(R2)=R2 δ(Db(R2))= λ
- Da(R)=S ⇒ R→aS δ(Da(R))= λ ⇒ R→a
  - R0 → aR1 ----------
  - R1 → aR1 ----------
  - R1 → bR2 R1 → b
  - R2 → bR2 R2 → b

44

Bibliografía

- Libro Básico 1 Bibliografía (AAM). Enrique Alfonseca Cubero, Manuel Alfonseca Cubero, Roberto Moriyón Salomón. Teoría de autómatas y lenguajes formales. McGraw-Hill (2007).
  Apartado 7.2

- Libro Básico 2 Bibliografía (HMU). John E. Hopcroft, Rajeev Motwani, Jeffrey D.Ullman. Introducción a la teoría de autómatas, lenguajes y computación (3ª edición). Ed, Pearson Addison Wesley.
  Tema 3

- Libro Básico 4 Bibliografía (AAM). Manuel Alfonseca, Justo Sancho, Miguel Martínez Orga. Teoría de lenguajes, gramáticas y autómatas. Publicaciones R.A.E.C. 1997
  Tema 7

45

5. Lenguajes Regulares

Grado Ingeniería Informática
Teoría de Autómatas y Lenguajes Formales

AUTÓMATAS FINITOS Y G3

2

Gramática asociada a un AF

Sea el AF, A = (Σ, Q, q0, f, F), existe una G3 LD tal que

L(G3LD) = L(A)

Es decir, el lenguaje que genera la gramática es el mismo que reconoce el Autómata

Veamos como se obtiene la gramática G={ΣT, ΣN, S, P} a partir del AF= {Q, Σ, q0, f, F}.

3

Gramática asociada a un AF

Se construye la **gramática G3LD** (G={Σ_T, Σ_N, S, P}) de la siguiente forma, **a partir del Autómata** (AF= {Σ, Q, q_0, f, F}):

- $\Sigma_T = \Sigma$; $\Sigma_N = Q$; $S = q_o$
- P= { … }
  1. transición f(p,a) = q → si q ∉ F→ **p::= a q**
  2. Si q ∈ F y f(p,a) = q → **p::= a** y **p::= a q**
  3. Si q_0 ∈ F → **q_0 ::= λ (es axioma para dar** λ)
  4. si f(p, λ) = q → si q ∉ F → p ::= q (redenominación);
  5. q ∈ F y f(p, λ) = q → p ::= q y p::= λ (redenominación y no generativa)

4

AF asociado a una G3 (cuando es LD)

De AF → G3: Ejemplo

Sea el AF descrito por la siguiente tabla, hallar la G3LD que genera el lenguaje por ella descrito. Comprobar que los lenguajes son iguales

	0	1
→A	A	C
B	A	C
C*	C	B

AF asociado a una G3LD

Sea la G3LD, G=(Σ_T, Σ_N S, P), existe un AF, A, tal que:

L(G3LD) = L(A)

Cómo se construye **AF a partir de G3LD**:

- $\Sigma = \Sigma_T$
- $Q = \Sigma_N \cup \{F\}$, con $F \notin \Sigma_N$
- $q_0 = S$
- $F = \{F\}$
- f:

  Si A ::= a B → f(A,a) = B

  Si A ::= a → f(A,a) = F

  Si S ::= λ → f(S, λ) = F (equivalente a hacer $q_0 = S \in F$)

AF asociado a una G3 (cuando es LD)

Se ha visto el procedimiento para obtener el AF que aceptaba el lenguaje descrito por una G3LD, sin embargo, ese procedimiento no siempre conduce a un AFD.

Lo habitual es: G3LD → AFND → AFD

1. Ejemplo: Sea la G3LD hallar el AF correspondiente.

G = ({d,c}, {A,S,T}, A, {A ::= cS, S::= d|cS|dT, T::= dT|d})

AF asociado a una G3

¿Y si queremos obtener un AF a partir de una G3LI?

G3LI → G3LD → AF

¿Y si queremos obtener una G3LI a partir de un AF?

AF → G3LD → G3LI

EXPRESIONES REGULARES

- Definición
- Expresiones y Lenguajes Regulares
- Equivalencias
- Teoremas

Definición de ER (I)

“ Metalenguaje para expresar el conjunto de palabras aceptadas por un AF (es decir, para expresar lenguajes de tipo 3 o regulares)”

Kleene, 1956

Definición de ER(I)

Ejemplo

Dado el alfabeto Σ= {0,1},

La ER 0*10* es una palabra del metalenguaje que representa las infinitas palabras del lenguaje regular formado por un 1, precedido y seguido de 0, 1 o infinitos 0s.

El lenguaje Σ* puede representarse mediante la ER:

(0+1)*

El lenguaje {01, 101} puede representarse mediante la ER:

01 + 101

La ER 1(1+0)* representa todas las cadenas que empiezan por el símbolo 1.

Definición de ER(II)

Dados los elementos/símbolos:

Σ, $\varnothing$ (lenguaje vacío), λ (palabra vacía)

y las operaciones:

$+$ (unión), $\bullet$(concatenación), $*$(cierre o clausura)

se cumple que:

- $\varnothing$ es una ER
- λ es una ER
- cualquier $a \in \Sigma$ es una ER
- si α y β son EERR entonces $\alpha+\beta$ y $\alpha\bullet\beta$ son EERR
- si α es una ER entonces α^* es una ER, donde $\alpha^* = \bigcup_{i=0}^{\infty} \alpha^i$

13

Definición de ER(III)

Solo son EERR las que se obtienen de aplicar las reglas anteriores **un número finito de veces** sobre símbolos de Σ, $\varnothing$, λ

La prioridad de las operaciones es la siguiente:

$$* > \bullet > +$$

14

- Definición
- Expresiones y Lenguajes Regulares
- Equivalencias
- Teoremas

EERR y LR

Cada Expresión Regular (ER) describe o expresa un lenguaje regular

A cada ER α, se le asocia un subconjunto de Σ^*, $L(\alpha)$, que es el LR descrito por α. Este lenguaje se define con:

- si $\alpha = \varnothing$, $L(\alpha) = \varnothing$
- si $\alpha = \lambda$, $L(\alpha) = \{\lambda\}$
- si $\alpha = a$, $a \in \Sigma$, $L(\alpha) = \{a\}$
- si α y β son EERR $\Rightarrow L(\alpha + \beta) = L(\alpha) \cup L(\beta)$
- si α y β son EERR $\Rightarrow L(\alpha \bullet \beta) = L(\alpha)\, L(\beta)$
- si α * es una ER $\Rightarrow L(\alpha^*) = L(\alpha)^*$

- Definición
- Expresiones y Lenguajes Regulares
- Equivalencias
- Teoremas

Equivalencia de EERR (I)

Dos EERR son equivalentes, $\alpha = \beta$, si describen el mismo lenguaje regular, si $L(\alpha) = L(\beta)$

Se cumple:

1) $(\alpha + \beta) + \sigma = \alpha + (\beta + \sigma)$ (+ es asociativa)
2) $\alpha + \beta = \beta + \alpha$ (+ es conmutativa)
3) $(\alpha \bullet \beta) \bullet \sigma = \alpha \bullet (\beta \bullet \sigma)$ (• es asociativa)
4) $\alpha \bullet (\beta + \sigma) = (\alpha \bullet \beta) + (\alpha \bullet \sigma)$ (+ es distributiva
   $(\beta + \sigma) \bullet \alpha = (\beta \bullet \alpha) + (\sigma \bullet \alpha)$ respecto de •)
5) $\alpha \bullet \lambda = \lambda \bullet \alpha = \alpha$ (• tiene elemento neutro)
6) $\alpha + \varnothing = \varnothing + \alpha = \alpha$ (+ tiene elemento neutro)
7) $\lambda^* = \lambda$
8) $\alpha \bullet \varnothing = \varnothing \bullet \alpha = \varnothing$

Equivalencia de EERR (II)

9) $\varnothing^* = \lambda$

10) $\alpha^* \bullet \alpha^* = \alpha^*$

11) $\alpha \bullet \alpha^* = \alpha^* \bullet \alpha$

12) $(\alpha^*)^* = \alpha^*$ (IMPORTANTE)

13) $\alpha^* = \lambda + \alpha + \alpha^2 + .. + \alpha^n + \alpha^{n+1}. \alpha^*$

14) $\alpha^* = \lambda + \alpha \bullet \alpha^*$ (13 con n=0) (IMPORTANTE)

15) $\alpha^* = (\lambda + \alpha)n\text{-}1 + \alpha n \bullet \alpha^*$ (de 14, sustituyendo)

16) Sea f una función, $f{:}E^n{}_\Sigma \rightarrow E_\Sigma$ se verifica:

$$f(\alpha, \beta, \ldots, \sigma) + (\alpha + \beta + \ldots + \sigma)^* = (\alpha + \beta + \ldots + \sigma)^*$$

17) Sea f una función, $f{:}E^n{}_\Sigma \rightarrow E_\Sigma$ se verifica:

$$(f(\alpha^*, \beta^*, \ldots, \sigma^*))^* = (\alpha + \beta + \ldots + \sigma)^*$$

19

Equivalencia de EERR (III)

18) $(\alpha^* + \beta^*)^* = (\alpha^* \bullet \beta^*)^* = (\alpha + \beta)^*$ (IMPORTANTE)

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

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

21) $(\alpha^* \bullet \beta)^* = \lambda + (\alpha + \beta)^* \bullet \beta$ (de 14 con 20)

22) Reglas de Inferencia:

Dadas tres EERR (L, A y B), sea la ecuación

L = AL + B,

donde $\lambda \notin A$, entonces se verifica que

L = A*B

20

- Definición
- Expresiones y Lenguajes Regulares
- Equivalencias
- Teoremas

Teoremas de análisis y síntesis de Kleene

1. Teorema de análisis de Kleene

Todo lenguaje aceptado por un AF es un lenguaje regular.

Solución al problema de análisis:

Encontrar el lenguaje asociado a un determinado AF: “Dado un AF, A, encontrar la ER que describe L(A)”.

1. Teorema de síntesis de Kleene

Todo lenguaje regular es el lenguaje aceptado por un AF.

Solución al problema de síntesis:

Encontrar un reconocedor para un lenguaje regular dado: *“Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular”.*

1. Solución al problema de análisis. Ecuaciones características.

Problema Análisis: AF -> ER

Resolución:

Dado un AF, escribir las ecuaciones características de cada uno de sus estados, resolverlas y obtener la ER buscada.

Solución al problema de análisis. Ecuaciones características.

ECUACIONES CARACTERÍSTICAS:

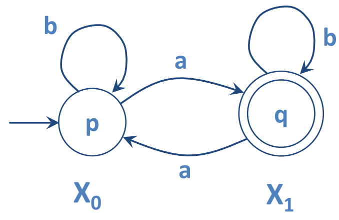
Describen todas las cadenas que se pueden reconocer desde un estado dado

Se escribe una ecuación x_i por estado q_i

- Primer miembro: $\mathbf{x_i}$
- El segundo miembro tiene un término por cada rama que salga de q_i
  - Las ramas tienen la forma $\mathbf{a_{ij} \bullet x_j}$ donde a_{ij} es la etiqueta de la rama que une q_i con q_j, x_j es la variable correspondiente a q_j
  - **Se añade un término a_{ij} por cada rama que une qi con un estado final**
  - **Se añade λ si q_i es final.**
  - Si de un estado q_i no sale ninguna rama, el segundo miembro será:
    - si es final: $x_i = \lambda$
    - si no es final: $x_i = \varnothing$

Solución al problema de análisis. Ecuaciones características.

Ecuaciones Características de un AF – Ejemplo 1:



El AF tiene 2 estados, por lo que tendrá 2 ecuaciones características:

Conjunto de palabras que permiten pasar desde el estado *p* a un estado final.

$X_0 = b\ X_0 + a\ X_1 + a$

Porque *q* es un estado final

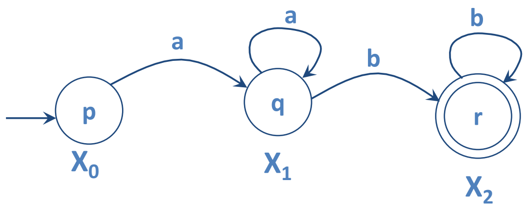
$X_1 = b\ X_1 + a\ X_0 + b$

Porque *q* es un estado final

25

Solución al problema de análisis. Ecuaciones características.

Ejemplo 2:



Ecuaciones Características:

$X_0 = a\ X_1$
$X_1 = b\ X_2 + a\ X_1 + b$
$X_2 = b\ X_2 + b$

26

Algoritmo de resolución problema de Análisis.

1. Escribir las ecuaciones características del AF
2. Resolverlas
3. Si el estado inicial es q_0, X_0 nos da el conjunto de cadenas que conducen desde q_0 a q_f y por tanto el lenguaje aceptado por el AF

27

Solución de las ecuaciones características.

La Ecuación Característica de la forma: $X = AX + B$, donde:

X: conjunto de cadenas que permiten pasar de q_i a $q_f \in F$

A: conjunto de cadenas que permiten, partiendo de un estado q, llegar a q.

B: conjunto de cadenas que permiten llegar al estado final, sin volver a pasar por el q_i de partida.

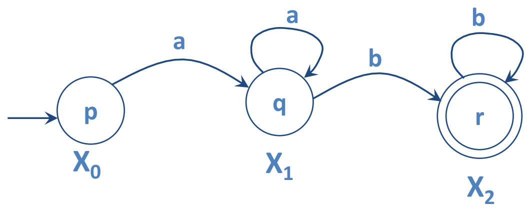
$\Downarrow$ (solución de Arden o reducción al absurdo)

La solución es: $\mathbf{X = A^* \bullet B}$

28

Solución al problema de análisis. Ecuaciones características.

Ejemplo 1:



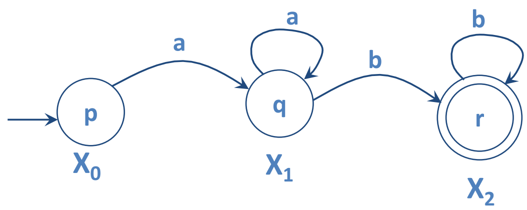
Ecuaciones Características:

$X_0 = a\,X_1$
$X_1 = b\,X_2 + a\,X_1 + b$
$X_2 = b\,X_2 + b + \lambda$

29

Solución al problema de análisis. Ecuaciones características.

Ejemplo 1:



Recuerda:

L=AL+B

L=A*B

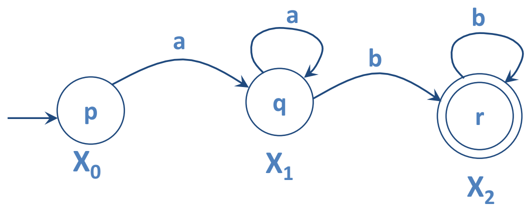
Ecuaciones Características:

$X_0 = a\,X_1$
$X_1 = b\,X_2 + a\,X_1 + b$
$X_2 = b\,X_2 + b + \lambda$

30

Solución al problema de análisis. Ecuaciones características.

Ejemplo 1:



Recuerda:
L=AL+B

L=A*B

Ecuaciones Características:

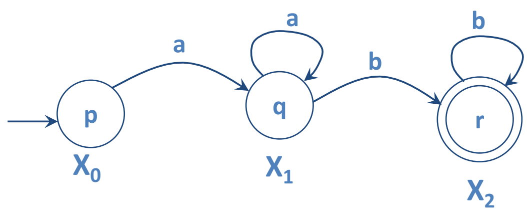
$X_0 = a\,X_1$
$X_1 = b\,X_2 + a\,X_1 + b$
$X_2 = b\,X_2 + b + \lambda$

$X_2 = b^* (b + \lambda)$

31

Solución al problema de análisis. Ecuaciones características.

Ejemplo 1:



Recuerda:
L=AL+B

L=A*B

Ecuaciones Características:

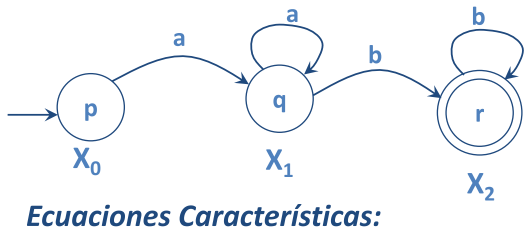
$X_0 = a\,X_1$
$X_1 = b\,X_2 + a\,X_1 + b$
$X_2 = b\,X_2 + b + \lambda$

$X_2 = b^* (b + \lambda)$
$X_1 = bb^* (b+ \lambda) + aX_1 + b$
$X_1 = aX_1 + bb^* (b+ \lambda) + b$
$X_1 = a^*(b\,b^* (b+ \lambda) + b) =$
$a^*b(b^*b+b^*+ \lambda) = a^*bb^*$

32

Solución al problema de análisis. Ecuaciones características.

Ejemplo 1:



Recuerda:
L=AL+B

L=A*B

Ecuaciones Características:

$X_0 = a\,X_1$
$X_1 = b\,X_2 + a\,X_1 + b$
$X_2 = b\,X_2 + b + \lambda$

$\cancel{X_2 = b^* (b + \lambda) = b^* + b^* = b^*}$
$\cancel{X_1 = bb^* + aX_1 + b}$
$\cancel{X_1 = aX_1 + bb^* + b}$
$\cancel{X_1 = a^*(bb^*+b) = a^*bb^*}$
$X_0 = aa^*bb^*$

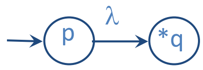
33

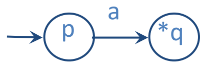
2. Problema de Síntesis: Algoritmo Recursivo (I)

Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular.

Sea α una Expresión Regular

- si $\alpha = \varnothing$, el autómata será: 

- si $\alpha = \lambda$, el autómata será: 

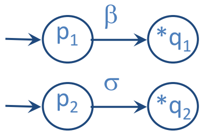
- si $\alpha =$ a, a$\in\Sigma$, el autómata será: 

34

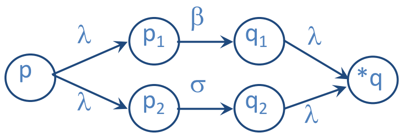
Problema de Síntesis: Algoritmo Recursivo (II)

Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular *(cont.)*

- si $\alpha=\beta+\sigma$, con los autómatas de β y σ



el resultado es:

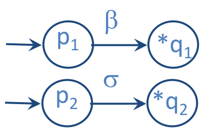


35

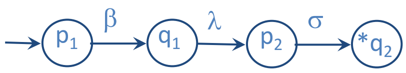
Problema de Síntesis: Algoritmo Recursivo (III)

Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular. (*cont.)*

- si $\alpha=\beta \bullet \sigma$, con los autómatas de β y σ



el resultado es:

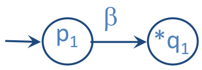


36

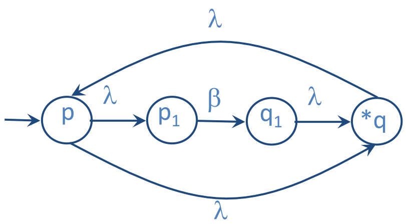
Problema de Síntesis: Algoritmo Recursivo (IV)

Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular. *(cont.)*

- si $\alpha=\beta^*$, con el autómata de β



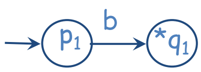
el resultado es:



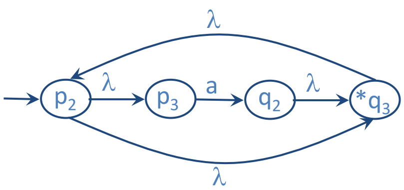
Problema de Síntesis: Algoritmo Recursivo (IV)

Ejemplo: Sea α= b • a*

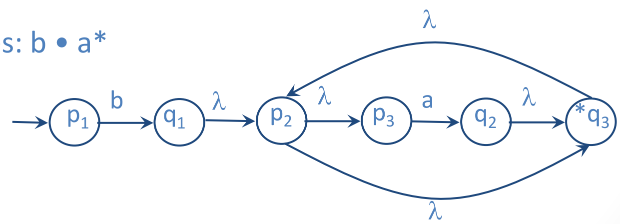
b:



a*:



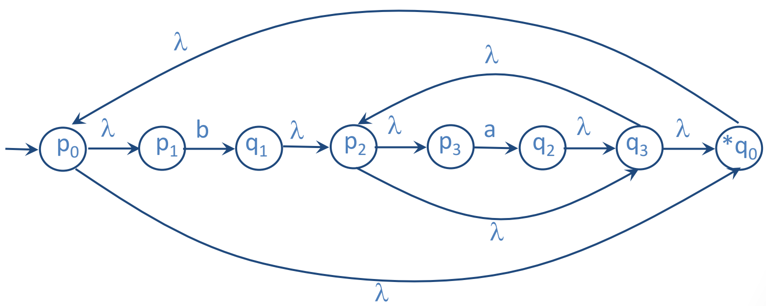
Entonces: b • a*



Problema de Síntesis: Algoritmo Recursivo (IV)

Ejemplo:

Sea $\alpha=(b \bullet a^*)^*$



39

2. Problema de Síntesis: Derivada de una ER.

Dada una ER, construir un AF que reconozca el lenguaje que la ER describe.

Solución: derivar la ER y obtener una G3LD y de ella un AF

Derivada de una ER: $D_a(R) = \{ x \mid a \bullet x \in R \}$.

- Derivada de ER R respecto de $a \in \Sigma$ es el conjunto de colas de todas las palabras representadas por R cuya cabeza es a.

Veamos una definición recursiva

40

Problema de Síntesis: Derivada de una ER

ER -> AF (Derivar la ER -> G3LD -> AF. $Da(R) = \{ x \mid a \bullet x \in R \}$

Derivada de una ER. ***Definición recursiva***

$\forall$ a, b $\in \Sigma$ y R,S expresiones regulares

- $Da(\varnothing) = \varnothing$
- $Da(\lambda) = \varnothing$
- $Da(a) = \lambda, \quad a \in \Sigma$
- $Da(b) = \varnothing, \quad \forall\, b \neq a,\, b \in \Sigma$
- $Da(R+S) = Da(R) + Da(S)$
- $Da(R \bullet S) = Da(R) \bullet S + \delta(R) \bullet Da(S) \quad \forall R$
  - $\lambda \in R \Rightarrow \delta(R) = \lambda$
  - $\lambda \notin R \Rightarrow \delta(R) = \varnothing$
- $Da(R^*) = Da(R) \bullet R^*$

41

Solución al problema de Síntesis. Derivada de una ER

- **Definición: Dab(R) = Db(Da(R))**
- A partir de la Derivada de una ER. Se obtendrá la gramática regular lineal derecha:
  - El número de derivadas distintas de una ER es finito.
    Una vez que se han obtenido todas, se puede obtener la G3
  - Sea $Da(R) = S$, con $S \neq \Phi$
    - $S \neq \lambda \Rightarrow R ::= aS \in P$
    - $S = \lambda \Rightarrow R ::= a \in P$
  - Sea $\delta(Da(R)) = S$
    - $\delta(Da(R)) = \lambda \Rightarrow R ::= a \in P$
    - $\delta(Da(R)) = \Phi \Rightarrow$ no se incluye ninguna regla en P
  - El axioma es R (ER de partida)
    - Σ_T = símbolos que formaban la ER de partida
    - Σ_N = letras que distinguen cada una de las derivadas distintas

42

Ejemplos. Derivada Expresiones Regulares

Obtener las G3 LD equivalentes a las ER dadas:

R = a a* b b*, Σ={a,b}

R = a a* b b* es igual que
R = a • a* • b • b*

- Da(R) = Da(a) a* b b* = a* b b*
- Db(R) = ∅
- Daa(R) = Da(a* b b*) = Da(a*) b b* + λ Da(b b*) = a*bb* = Da(R)
- Dab(R) = Db(a* b b*) = Db(a*) b b* + λ Db(b b*) = b*
- Daba(R) = Da(b*) = ∅
- Dabb(R) = Db(b*) = Db(b) b* = b* = Dab(R)

- Da(R)= a*bb* δ(Da(R))= ∅
- Daa(R)= a*bb* δ(Daa(R))= ∅
- Dab(R)= b* δ(Dab(R))= λ
- Dabb(R)= b* δ(Dabb(R))= λ

43

Ejemplos. Derivada Expresiones Regulares

- Obtención de la G3 LD que genera el Lenguaje representado por la ER:
- R0=aa*bb* R1=a*bb* R2=b*
  - Da(R0)=R1 δ(Da(R0))= ∅
  - Da(R1)=R1 δ(Da(R1))= ∅
  - Db(R1)=R2 δ(Db(R1))= λ
  - Db(R2)=R2 δ(Db(R2))= λ
- Da(R)=S ⇒ R→aS δ(Da(R))= λ ⇒ R→a
  - R0 → aR1 ----------
  - R1 → aR1 ----------
  - R1 → bR2 R1 → b
  - R2 → bR2 R2 → b

44

Bibliografía

- Libro Básico 1 Bibliografía (AAM). Enrique Alfonseca Cubero, Manuel Alfonseca Cubero, Roberto Moriyón Salomón. Teoría de autómatas y lenguajes formales. McGraw-Hill (2007).
  Apartado 7.2

- Libro Básico 2 Bibliografía (HMU). John E. Hopcroft, Rajeev Motwani, Jeffrey D.Ullman. Introducción a la teoría de autómatas, lenguajes y computación (3ª edición). Ed, Pearson Addison Wesley.
  Tema 3

- Libro Básico 4 Bibliografía (AAM). Manuel Alfonseca, Justo Sancho, Miguel Martínez Orga. Teoría de lenguajes, gramáticas y autómatas. Publicaciones R.A.E.C. 1997
  Tema 7

45

Parte VI

TEMA 6. Autómatas a Pila

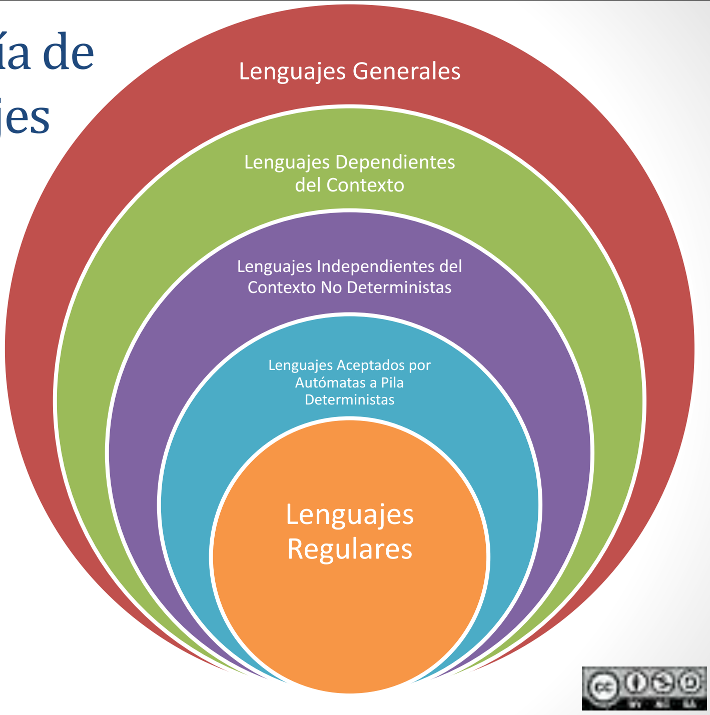
6. Autómatas a Pila

Grado Ingeniería Informática
Teoría de Autómatas y Lenguajes Formales

- Introducción
- Definición
- Equivalencias
- Lenguajes Tipo 2

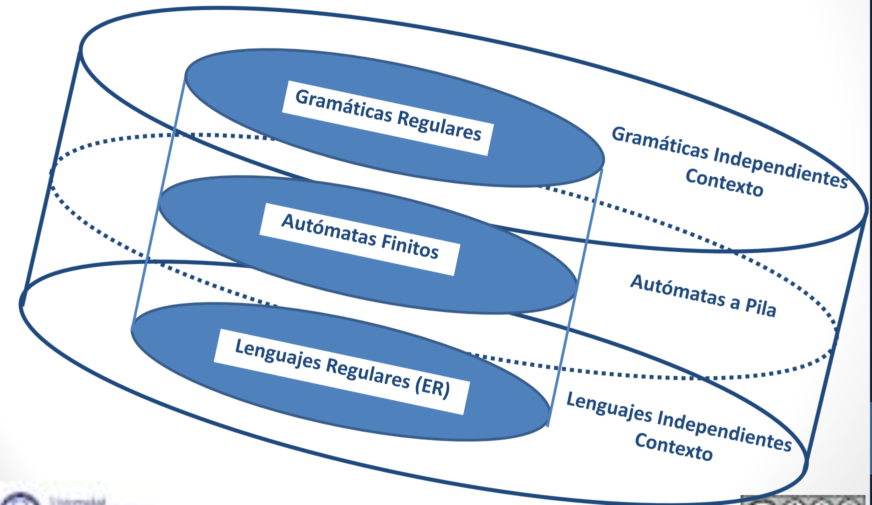
2

Jerarquía de Lenguajes



3

Jerarquía de Lenguajes

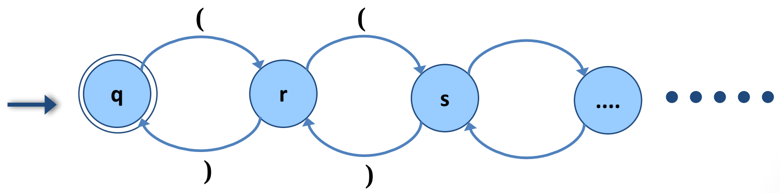


4

Limitaciones de los AF

Falta de memoria

No pueden reconocer expresiones matemáticas,
pej. (2x+(2+n/25)), mas general el lenguaje X^nY^n



5

Teoremas

1 Para cada gramática G independiente del contexto, existe un autómata de pila M tal que

L(G)=L(M)

2 Para cada autómata de pila M, existe una gramática G independiente del contexto tal que

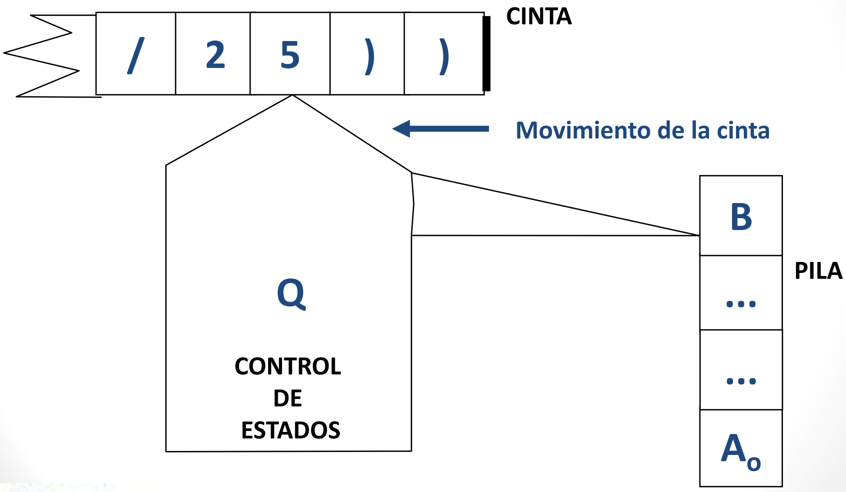
L(M)=L(G)

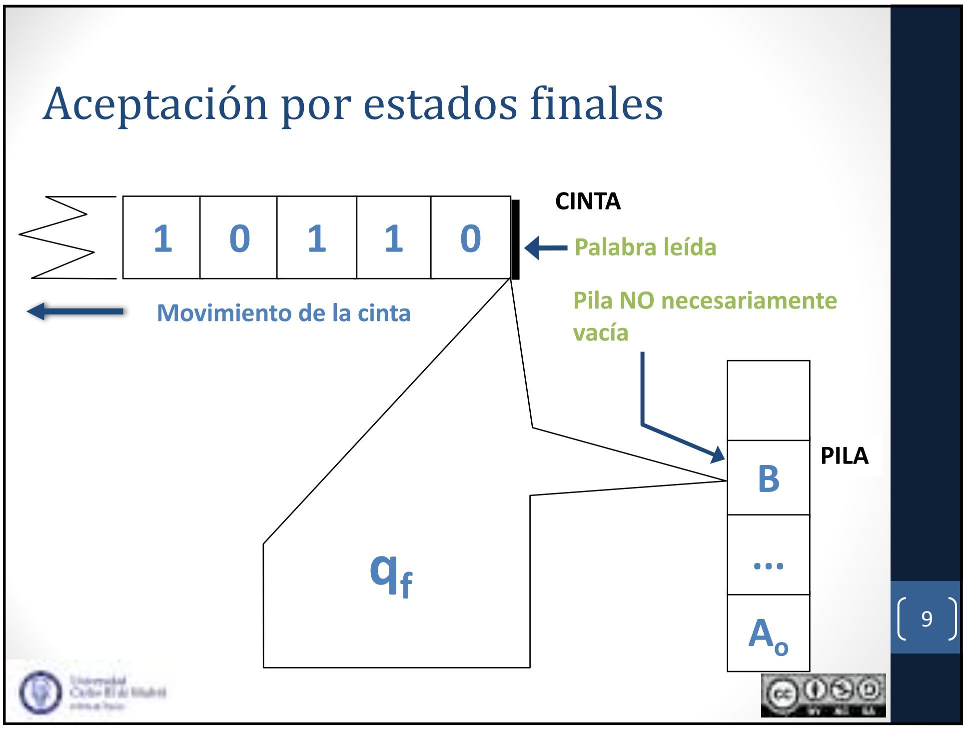
3 Existe un lenguaje independiente del contexto que no es el lenguaje aceptado por ningún autómata de pila determinista

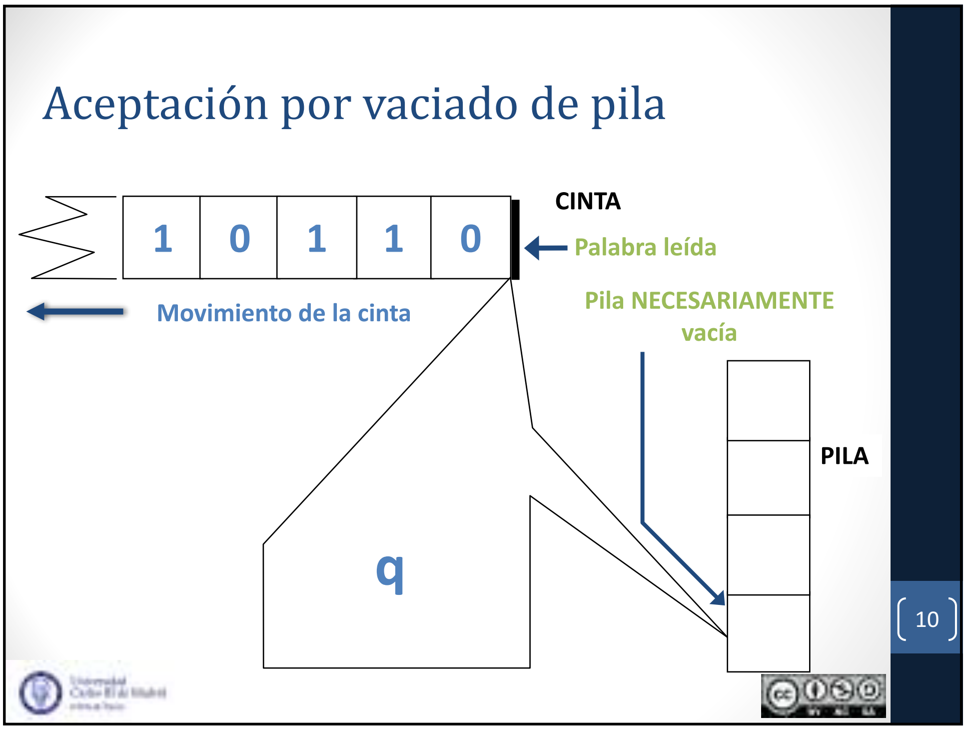
6

- Introducción
- Definición
- Equivalencias
- Lenguajes Tipo 2

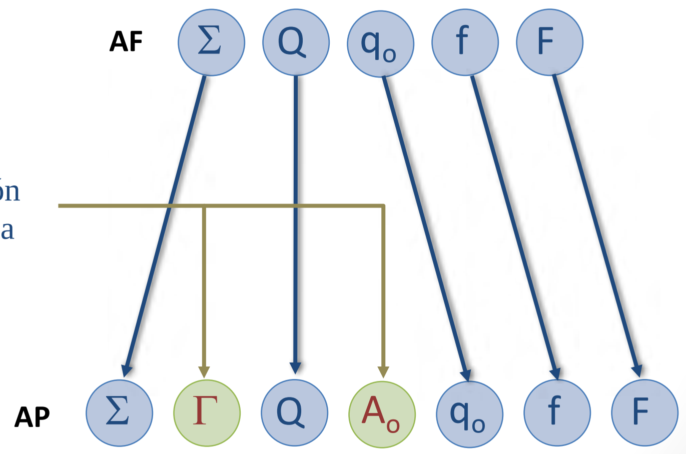
Definición de AP




Aceptación por estados finales
CINTA
1 0 1 1 0
Palabra leída
Movimiento de la cinta
Pila NO necesariamente vacía
PILA
B
…
A_o
q_f
9


Aceptación por vaciado de pila
CINTA
1 0 1 1 0
Palabra leída
Movimiento de la cinta
Pila NECESARIAMENTE vacía
PILA
q
10

Definición formal



11

AP: $(\Sigma, \Gamma, Q, A_o, q_o, f, F)$

Σ : alfabeto de entrada — Palabras: x, y, z, ax, ay...

Γ : alfabeto de pila — Palabras: X, Y, Z, AX, AY...

Q : conjunto finito de estados — Q = {p,q,r,...}

$A_o \in \Gamma$: símbolo inicial de la pila

$q_o \in Q$: estado inicial del autómata

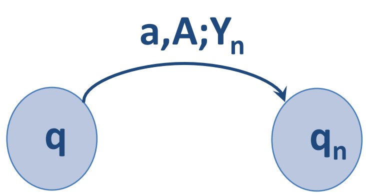
f : función de transición

$F \subset Q$: conjunto de estados finales

12

Transición

$f(q,a,A)=\{(q_1,Z_1),(q_2, Z_2),\ldots,(q_n, Z_n)\}$



1. Leer un símbolo de la entrada
2. Extraer un símbolo de la pila
3. Insertar una palabra en la pila
4. Pasar a un nuevo estado

$(q,a,A;q_n,Y_n)$

13

Función de Transición

$$f : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$$

Transiciones dependientes de la entrada

$Q \times \Sigma \times \Gamma$

Transiciones independientes de la entrada

$Q \times \lambda \times \Gamma$

AP Deterministas

$Q \times \Gamma^*$

AP No Deterministas

$P(Q \times \Gamma^*)$

14

T. independientes de la entrada

Sea la transición:

$f(q, \lambda, A) = \{(q_1, Z_1), (q_2, Z_2), \ldots, (q_n, Z_n)\}$

donde:

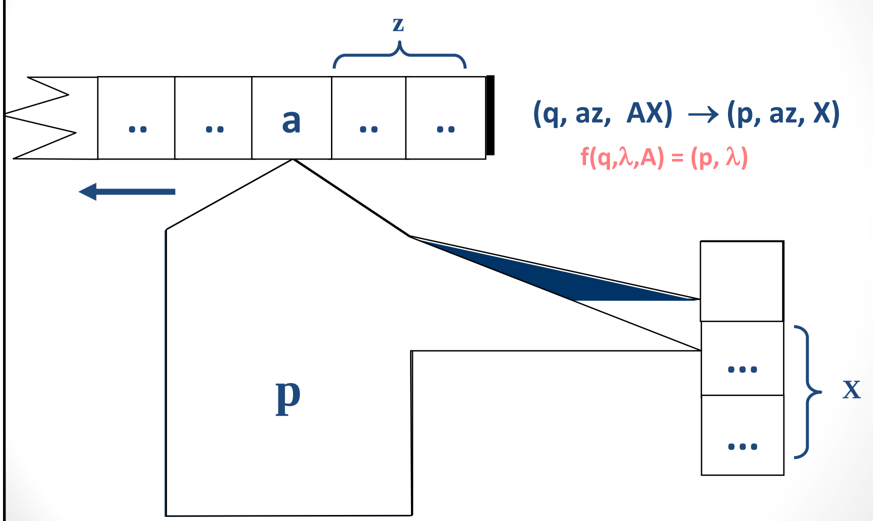
$q, q_i \in Q$

$A \in \Gamma$

$Z_i \in \Gamma^*$

15

T. independientes de la entrada



16

T. dependientes de la entrada

Sea la transición:

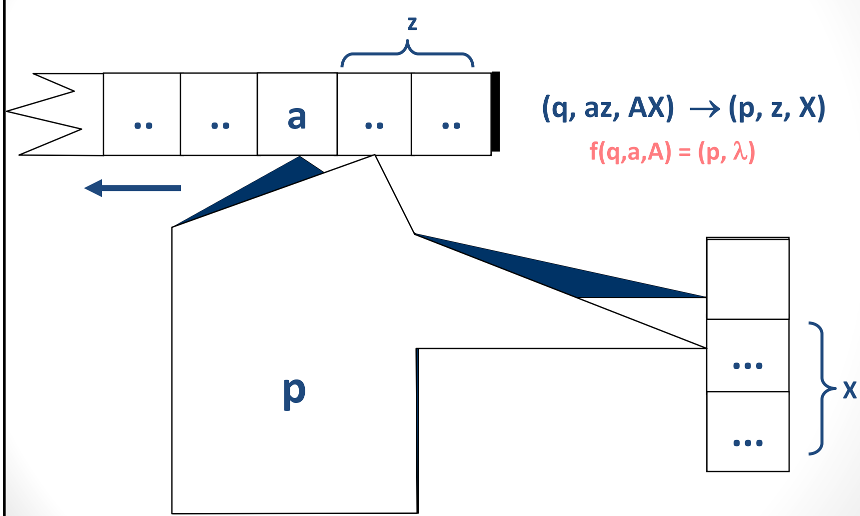
$f(q,a,A) = \{(q_1,Z_1), (q_2,Z_2),\ldots,(q_n,Z_n)\}$

donde:

- $q, q_i \in Q$
- $a \in \Sigma$
- $A \in \Gamma$
- $Z_i \in \Gamma^*$

17

T. dependientes de la entrada



$(q, az, AX) \rightarrow (p, z, X)$

$f(q,a,A) = (p, \lambda)$

18

Descripción Instantánea

Permite describir sencillamente la configuración del AP en cada momento

Terna (q,x,z) donde:

$$q \in Q,\ x \in \Sigma^*,\ z \in \Gamma^*$$

Contiene:
- el estado actual (**q**)
- lo que queda por leer de la entrada (**x**) y
- **el contenido de la pila (z) en un momento dado**

19

Descripción Instantánea

Movimiento: $(q,ay,AX) \vdash (p,y,YX)$
describe el paso de una descripción instantánea a otra

Sucesión de movimientos:
$(q,ay,AX) \vdash^* (p,y,YX)$
representa que desde la primera descripción instantánea se puede alcanzar la segunda

20

Autómatas a Pila Deterministas

$(\Sigma,\Gamma,Q,A_0,q_0,f,F)$ es determinista si verifica:

$\forall q\in Q,\ A\in\Gamma,\ |f(q,\lambda,A)|>0 \Rightarrow f(q,a,A)=\Phi\ \forall a\in\Sigma$

$\forall q\in Q,\ A\in\Gamma,\ \forall a\in\Sigma\cup\{\lambda\},\ |f(q,a,A)|<2$

Si $\exists\ f(q,\lambda,A)\ \nexists\ f(q,a,A)$ o
Si $\exists\ f(q,a,A)\ \nexists\ f(q,\lambda,A)$

si (p, x, y; q, z) y (p, x, y; r, w) son transiciones de un autómata a pila determinista entonces

$q\equiv r,\ z=w$

21

Lenguaje aceptado por un AP

Por vaciado de pila

$LV_{AP}=\{x\,|\,(q_0,x,A_0)\vdash^*(p,\lambda,\lambda),\ p\in Q,\ x\in\Sigma^*\}$

Por estado final

$LF_{AP}=\{x\,|\,(q_0,x,A_0)\vdash^*(p,\lambda,X),\ p\in F,\ x\in\Sigma^*,X\in\Gamma^*\}$

22

Ejemplo Alfonseca (viejo pp 224-225)

Dado el AP_V = ({0,1,2}, {A,0,1}, {p,q},A, p,f ϕ)

f(p,0,A) = (p,0A) ← Transiciones de inicio (implican a A)

f(p,0,0) = (p,00)

f(p,0,1) = (p,01) ← Transiciones de apilamiento (leo 0 y recuerdo 00 o 01)

f(p,1,A) = (p,1A) ← Transiciones de inicio (implican a A)

f(p,1,0) = (p,10)

f(p,1,1) = (p,11) ← Transiciones de apilamiento (leo 1 y recuerdo 11 o 10)

f(p,2,A) = (q,A) ← Transiciones de inicio (implican a A)

f(p,2,0) = (q, 0)

f(p,2,1) = (q, 1) ← Transiciones de cambio de estado No afecta a la pila

f(q,0,0) = (q, λ)

f(q,1,1) = (q, λ)

f(q, λ,A) = (q, λ) ← Transiciones de desapilamiento (leo 0 o 1 y apilo λ)

23

Ejemplo Alfonseca (viejo pp 224-225)

Es un AP por estado final

LF_{AP}={w 2 w^{-1}, w$\in$(0+1)*}

24

Ejemplo

LENGUAJE: algunas instrucciones

var ::= num; (asignación)

if cond
then
BLOQUE (asignación ó if)

if cond
then
BLOQUE (asignación ó if)
else
BLOQUE (asignación ó if)

25

Ejemplo

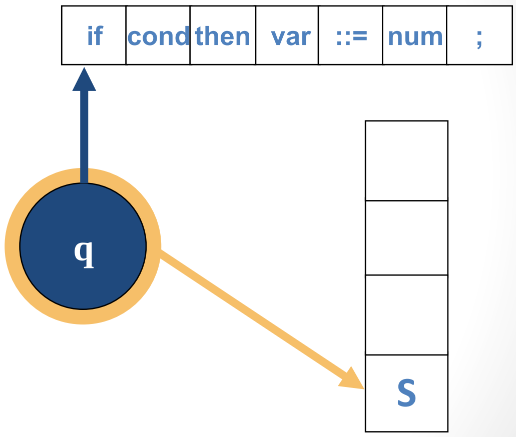
AP= ({if, then, else, ::=, var, num, cond, ;}, {S, B, C, F, N, P, T, E}, {q}, q, S, f, ϕ)

f(q, var, S) = {(q, FNP)}
f(q, if, S) = {(q, CTBP), (q, CTBEBP)}
f(q, if, B) = {(q, CTB), (q, CTBEB)}
f(q, var, B) = {(q, FN)}
f(q, cond, C) = {(q, λ)}
f(q, ::=, F) = {(q, λ)}
f(q, num, N) = {(q, λ)}
f(q, ;, P) = {(q, λ)}
f(q, then, T) = {(q, λ)}
f(q, else, E) = {(q, λ)}

ELEMENTOS DE Γ
- S → Símbolo inicial
- F → ::=
- N → Numero
- P → ;
- C → Condición
- T → Then
- B → Bloque
- E → Else

26

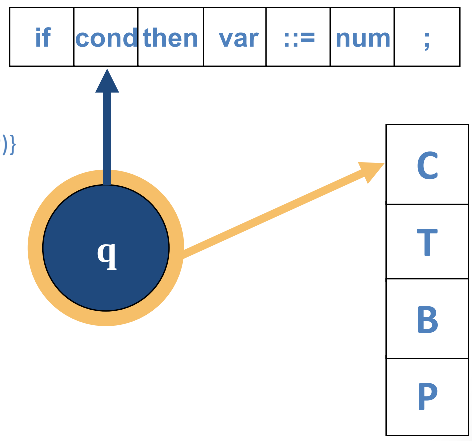
Ejemplo



f(q, var, S) = {(q, FNP)}
f(q, if, S) = {(q, CTBP), (q, CTBEBP)}
f(q, if, B) = {(q, CTB), (q, CTBEB)}
f(q, var, B) = {(q, FN)}
f(q, cond, C) = {(q, λ)}
f(q, ::=, F) = {(q, λ)}
f(q, num, N) = {(q, λ)}
f(q, ;, P) = {(q, λ)}
f(q, then, T) = {(q, λ)}
f(q, else, E) = {(q, λ)}

27

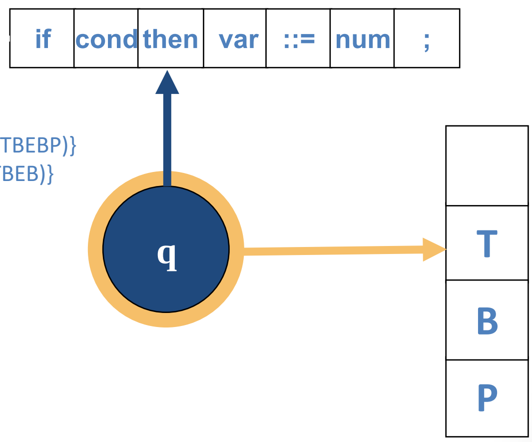
Ejemplo



f(q, var, S) = {(q, FNP)}
f(q, if, S) = {(q, CTBP), (q, CTBEBP)}
f(q, if, B) = {(q, CTB), (q, CTBEB)}
f(q, var, B) = {(q, FN)}
f(q, cond, C) = {(q, λ)}
f(q, ::=, F) = {(q, λ)}
f(q, num, N) = {(q, λ)}
f(q, ;, P) = {(q, λ)}
f(q, then, T) = {(q, λ)}
f(q, else, E) = {(q, λ)}

28

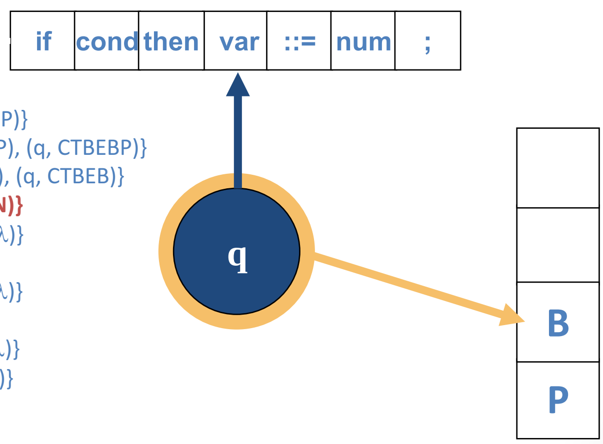
Ejemplo



f(q, var, S) = {(q, FNP)}
f(q, if, S) = {(q, CTBP), (q, CTBEBP)}
f(q, if, B) = {(q, CTB), (q, CTBEB)}
f(q, var, B) = {(q, FN)}
f(q, cond, C) = {(q, λ)}
f(q, ::=, F) = {(q, λ)}
f(q, num, N) = {(q, λ)}
f(q, ;, P) = {(q, λ)}
f(q, then, T) = {(q, λ)}
f(q, else, E) = {(q, λ)}

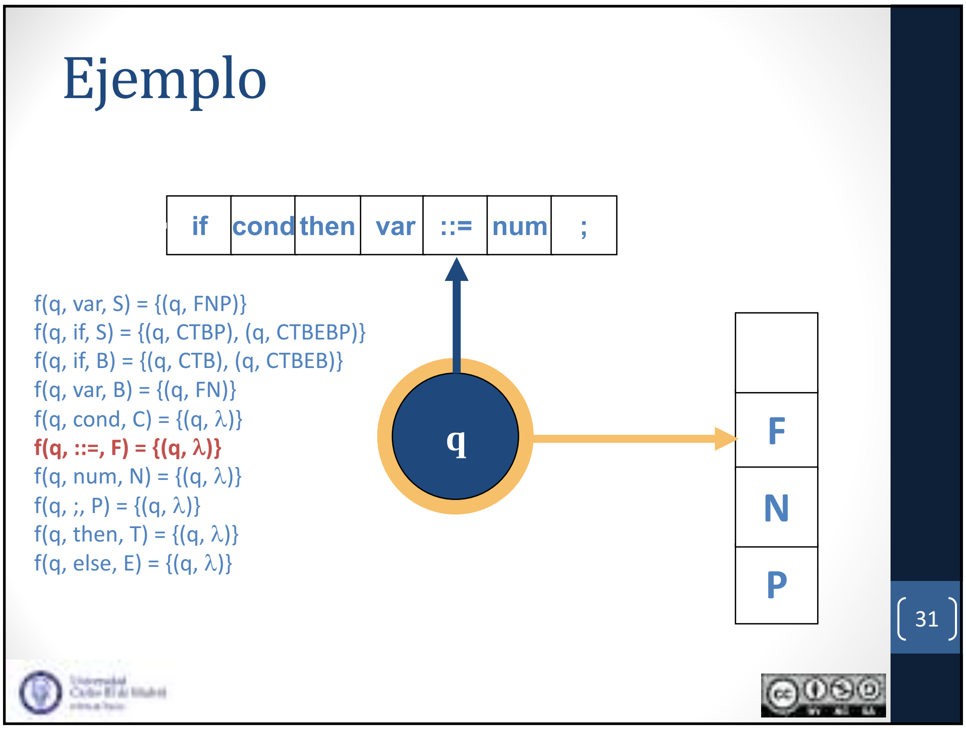
29

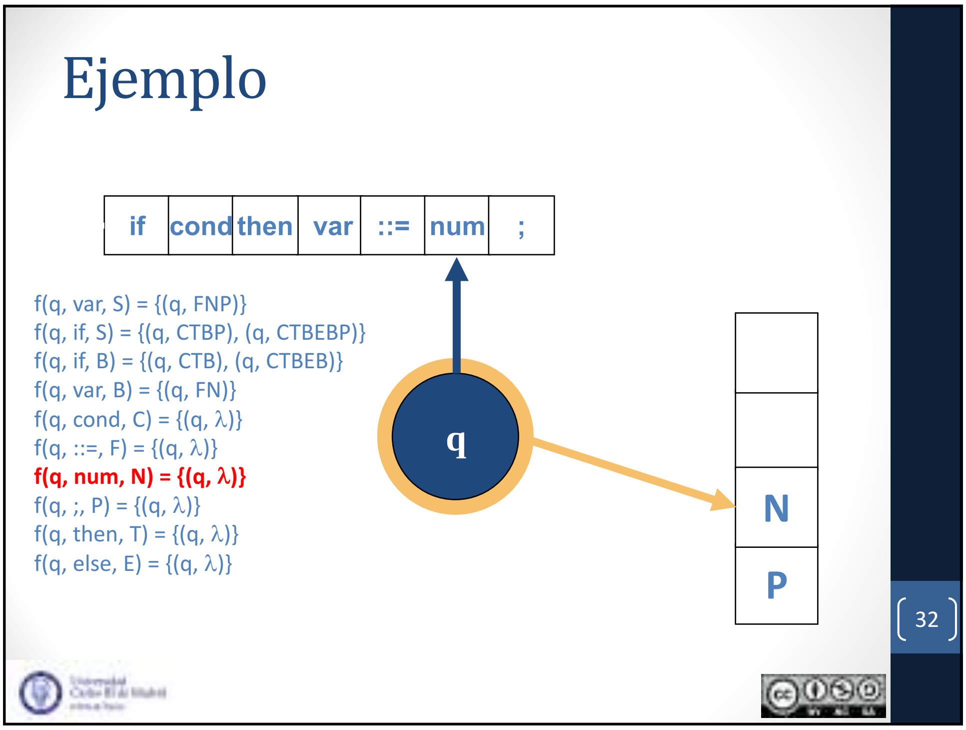
Ejemplo



f(q, var, S) = {(q, FNP)}
f(q, if, S) = {(q, CTBP), (q, CTBEBP)}
f(q, if, B) = {(q, CTB), (q, CTBEB)}
f(q, var, B) = {(q, FN)}
f(q, cond, C) = {(q, λ)}
f(q, ::=, F) = {(q, λ)}
f(q, num, N) = {(q, λ)}
f(q, ;, P) = {(q, λ)}
f(q, then, T) = {(q, λ)}
f(q, else, E) = {(q, λ)}

30

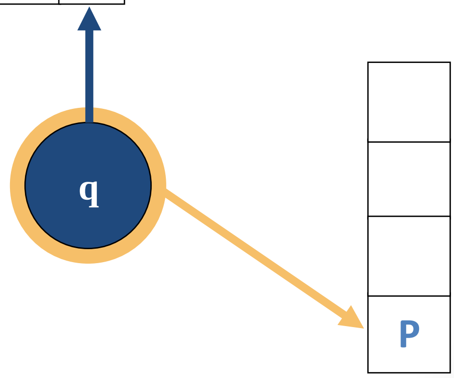

Ejemplo
if
cond
then
var
::=
num
;
f(q, var, S) = {(q, FNP)}
f(q, if, S) = {(q, CTBP), (q, CTBEBP)}
f(q, if, B) = {(q, CTB), (q, CTBEB)}
f(q, var, B) = {(q, FN)}
f(q, cond, C) = {(q, λ)}
f(q, ::=, F) = {(q, λ)}
f(q, num, N) = {(q, λ)}
f(q, ;, P) = {(q, λ)}
f(q, then, T) = {(q, λ)}
f(q, else, E) = {(q, λ)}
q
F
N
P
31


Ejemplo
if
cond
then
var
::=
num
;
f(q, var, S) = {(q, FNP)}
f(q, if, S) = {(q, CTBP), (q, CTBEBP)}
f(q, if, B) = {(q, CTB), (q, CTBEB)}
f(q, var, B) = {(q, FN)}
f(q, cond, C) = {(q, λ)}
f(q, ::=, F) = {(q, λ)}
f(q, num, N) = {(q, λ)}
f(q, ;, P) = {(q, λ)}
f(q, then, T) = {(q, λ)}
f(q, else, E) = {(q, λ)}
q
N
P
32

Ejemplo

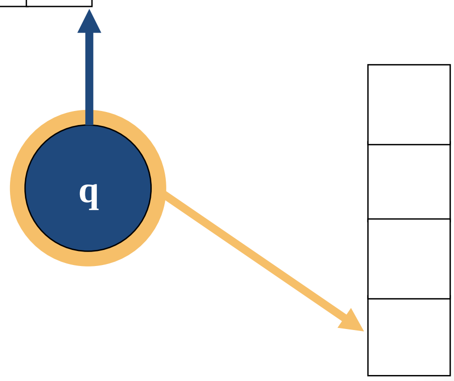
if	cond	then	var	::=	num	;

f(q, var, S) = {(q, FNP)}
f(q, if, S) = {(q, CTBP), (q, CTBEBP)}
f(q, if, B) = {(q, CTB), (q, CTBEB)}
f(q, var, B) = {(q, FN)}
f(q, cond, C) = {(q, λ)}
f(q, ::=, F) = {(q, λ)}
f(q, num, N) = {(q, λ)}
f(q, ;, P) = {(q, λ)}
f(q, then, T) = {(q, λ)}
f(q, else, E) = {(q, λ)}



Ejemplo

if	cond	then	var	::=	num	;



Pila vacía
Sentencia reconocida

- Introducción
- Definición
- Equivalencias
- Lenguajes Tipo 2

Equivalencias

Teorema

Para cada autómata de pila que acepte cadenas sin vaciar su pila, existe un autómata equivalente pero que vacía su pila antes de llegar a un estado de aceptación.

Paso de AP_F a AP_V

$AP_F = (\Sigma, \Gamma, Q, A_0, q_0, f, F)$

$AP_V = (\Sigma, \Gamma \cup \{B\}, Q \cup \{p,r\}, B, p, f', \phi)$

1. Nuevo símbolo para la pila
2. Dos estados nuevos
3. Valor inicial de la pila
4. Nuevo estado inicial
5. SIN estados finales

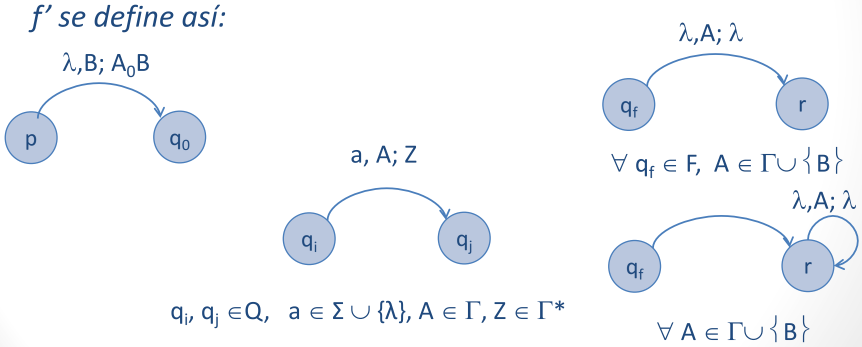
37

Paso de AP_F a AP_V

$AP_F = (\Sigma, \Gamma, Q, A_0, q_0, f, F)$

$AP_V = (\Sigma, \Gamma \cup \{B\}, Q \cup \{p,r\}, B, p, f', \phi)$

f' se define así:

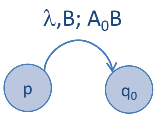


38

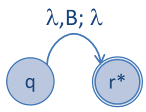
Paso de AP_V a AP_F

$AP_V=(\Sigma, \Gamma, Q, A0, q0, f, \phi) \rightarrow AP_F=(\Sigma, \Gamma \cup \{B\}, Q \cup \{p, r\}, B, p, f', \{r\})$

f' se define así:



$f'(p, \lambda, B) = (q_0, A_0B)$



$(r, \lambda) \in f'(q, \lambda, B) \quad \forall\, q \in Q,$

$f(q, a, A) = f'(q, a, A) \quad \forall\, q \in Q,\ a \in \Sigma \cup \{\lambda\}, A \in \Gamma$

- Introducción
- Definición
- Equivalencias
- Lenguajes Tipo 2

40

De Gramática Tipo 2 a AP_V

Dada una G2 en FNG, construir un AP_V:

$$G = (\Sigma_T, \Sigma_N, S, P)$$

entrada pila inicial de pila

$$AP_V = (\Sigma_T, \Sigma_N, \{q\}, S, q, f, \phi)$$

Se obtiene un AP_V con un solo estado

De Gramática Tipo 2 a AP_V

f se define como:

$$(q, Z) \in f(q, a, A)$$

es decir:

$f(q, a, A) = (q, Z)$ si existe una producción del tipo A ::= a Z

$f(q, a, A) = (q, \lambda)$ si existe una producción del tipo A ::= a

$f(q, a, A) = \{(q, Z), (q, \lambda)\}$

dada una producción:

A::= aZ | aD | b $\Rightarrow$ $f(q, a, A)= \{(q, Z), (q, D)\}$

$f(q, b, A) = (q, \lambda)$

Si S::= $\lambda \Rightarrow (q,\lambda) \in f(q,\lambda,S)$

De AP_V a Gramática Tipo 2

Dado un AP_V, construir una G2 tal que $L(G2) = L(AP_V)$

$$AP_V = (\Sigma, \Gamma, Q, A_0, q_0, f, \phi)$$

$$\Downarrow$$

$$G = (\Sigma_T, \underbrace{\Sigma_N}_{\{S\} \cup \{(pAq) \mid p,\, q \in Q,\ A \in \Gamma\}}, S, P)$$

Para construir P:

1. $S ::= (q_0, A_0, q)\ \forall\, q \in Q$ (se eligen las que empiezan por q_0A_0)
2. De cada transición $f(p,a,A) = (q, BB'B''....B''')$
   donde:
   $A,B,B',B'',...,B''' \in \Gamma$; $a \in \Sigma \cup \{\lambda\}$
   se obtiene:
   $(p\ A\ z) ::= a\ (q\ B\ r)\ (r\ B'\ s)\ s \ldots y\ (y\ B'''\ z)$
3. De cada transición $f(p, a, A) = (q, \lambda)$ se obtienen: $(p,A,q) ::= a$

45

Bibliografía

- Libro Básico 1 Bibliografía. Enrique Alfonseca Cubero, Manuel Alfonseca Cubero, Roberto Moriyón Salomón. Teoría de autómatas y lenguajes formales. McGraw-Hill (2007).
  Capítulo 4 y Apartado 8.1
- Libro Básico 2 Bibliografía. John E. Hopcroft, Rajeev Motwani, Jeffrey D.Ullman. Introducción a la teoría de autómatas, lenguajes y computación (3ª edición). Ed, Pearson Addison Wesley.
  Capítulo 6
- Libro Básico 4 Bibliografía. Manuel Alfonseca, Justo Sancho, Miguel Martínez Orga. Teoría de lenguajes, gramáticas y autómatas. Publicaciones R.A.E.C. 1997
  Capítulo 10

46

Parte VII

TEMA 7. Maquina de Turing

7. Máquinas de Turing

Grado Ingeniería Informática
Teoría de Autómatas y Lenguajes Formales

- Introducción, Definición y Ejemplos
- Tipos de MT
- Equivalencia y Variantes de MT

Introducción

Origen:

- La Máquina de Turing (MT) fue descrita por Alan Turing en 1936.

Alan Turing (Inglés: 1912 - 1956)

Fue un científico inglés que hizo grandes aportaciones en: matemáticas, criptoanálisis, lógica, filosofía, biología, ciencias de la computación, inteligencia artificial y vida artificial.

Es considerado uno de los padres de la ciencia de la computación. Es el precursor de la informática moderna.

3

Introducción

- "Propongo considerar la siguiente cuestión:¿pueden pensar las máquinas?". Ha nacido la IA.
- Test de Turing

Turing:http://www.youtube.com/watch?feature=player_embedded&v=ChJVaTqU2So

4

Introducción

Test de Turing:

- 1990, auspiciado por el filántropo Hugh Loebner, se celebra el **Premio Loebner** para honrar la memoria de Alan Turing.
- Una vez al año, robots de todo el mundo compiten para tratar de superar la prueba, aunque nadie ha podido hacerlo todavía.
- Si bien existen premios menores de consolación, los 100.000 dólares del premio gordo permanecen desiertos.
- Cuando un androide sea capaz de hacerse pasar por un humano antes los jueces, el concurso de desconvocará para siempre.
- Muchos han estado cerca, pero no ha llegado el día en que una máquina haya **imitado** la inteligencia humana.

5

Introducción

- Test de Turing:
  - 2010, cuando Suzette, de Bruce Wilcox, que volvió a ganar el certamen el año siguiente, logró engañar al juez del Premio Loebner. Una conversación sobre política en la que el *bot* sembró la confusión con una **imitación** casi perfecta de un humano.
  - En la palabra **imitar** está la clave. Los críticos del test de Turing esgrimen el argumento de que la mayoría de los robots que se enfrentan a él no están basados en un auténtica inteligencia artificial, sino en una especialmente dirigida para superar la prueba. Además, todo está permitido, desde las respuestas absurdas hasta las mentiras.
  - Wilcox es especialista en programación de *chatbots*, y su aplicación Tom Loves Angela, es ya un clásico de los programas de conversación basados en inteligencia artificial.

6

Introducción

- Test de Turing:
  - Según los expertos, el Premio Loebner es la prueba más fiel al test original, aunque existen una serie de concursos en la misma línea. Entre ellos, **Turing 100**, donde hace unos años se produjo una sorpresa protagonizada por un robot que imitaba a un adolescente de trece años.
  - El programa Eugene Goostman, también obtuvo los mejores resultados en varias ediciones del Premio Loebner, obtuvo un 29% de respuestas consideradas humanas, cuando el límite para superar el test de Turing se encuentra en el 30%.

7

Introducción

- Test de Turing: **Fecha límite**
  - Como ocurre con la ley de Moore, sobre el test de Turing existen especulaciones sobre cuándo los robots superarán el límite de su inteligencia.
  - Ray Kurzweil, ha predicho que un ordenador pasará la prueba de Turing en el año 2029, basándose en el concepto de la singularidad tecnológica.
  - En la misma línea, Luis Von Ahn, uno de los mayores expertos en inteligencia artificial de nuestra época: "*es difícil saberlo, pero quizás hasta dentro de 10 o 20 años no sea posible. Por ejemplo, un ordenador capaz de escribir noticias correctamente estaría muy cerca de superar el test de Turing*", explica el experto en ciencia computacional.

8

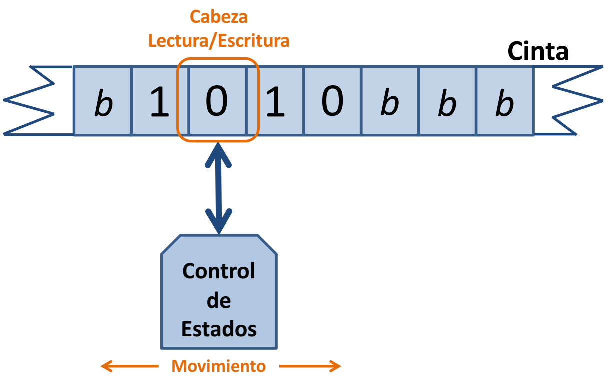
Definición de una MT

- Dispositivo hipotético capaz de manipular símbolos en una tira de cinta considerando ciertas reglas. A pesar de su simplicidad, pueden simular la lógica de cualquier algoritmo de un computador.
- Una MT está formado por:
  - Cinta infinita dividida en celdas
  - Cabezal de lectura/escritura capaz de moverse sobre dicha cinta.

Diferentes versiones que simulan una MT

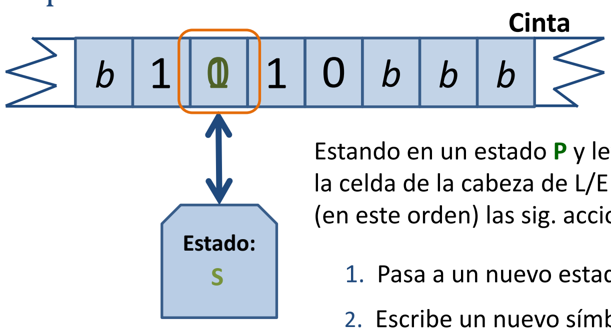
Definición de una MT

Representación:



Definición de una MT

Operaciones:



Estando en un estado **P** y leyendo el símbolo de la celda de la cabeza de L/E (Ej: 0), se realizan (en este orden) las sig. acciones:

1. Pasa a un nuevo estado. (Ej: S)
2. Escribe un nuevo símbolo en la cinta (reemplazando el existente). (Ej: 0 → 1)
3. Mueve el cabezal de L/E hacia: Drcha, Izqda, o no se mueve (Ej: Izqda)

11

Definición de una MT

Definición Formal:

Una MT se define como una 7-tupla:

$$\text{MT} = (\Sigma, \Gamma, b, Q, q_0, f, F)$$

Donde:

Símbolo	
$\Sigma \subset \Gamma$	Alfabeto de entrada.
Γ	Alfabeto de símbolos de la cinta.
$b \in \Gamma$	Símbolo especial - espacio en blanco ($b \not\subset \Sigma$). También se representa como: □
Q	Conjunto finito de estados.
$q_0 \in Q$	Estado inicial.
f	Función $Q \times \Gamma \rightarrow Q \times \Gamma \times$ {L(-),R(+),S(=)} (donde *L: Left*, *R: Right* y *S: Stay*).
$F \subseteq Q$	Conjunto de estados finales o de aceptación.

12

Definición de una MT

Características:

- La cinta se supone infinita por ambos lados.
- Inicialmente la cinta contiene un número finito de símbolos consecutivos (de Σ) precedidos y seguidos por el símbolo b (o □).
- La cabecera de L/E está situada inicialmente sobre el elemento más a la izquierda de la palabra.
- Toda MT se representa por una tabla de transición (como el resto de Autómatas). **Si la transición No es posible → La MT *se detiene*.**

f (Estados)	Símbolo	Símbolo	...
Estado	*(Estado, Símbolo, Movim.)*	*(Estado, Símbolo, Movim.)*	...
Estado	*(Estado, Símbolo, Movim.)*	*(Estado, Símbolo, Movim.)*	...
...	...	...	...

13

Ejemplo de una MT

$MT\_1 = (\boldsymbol{\Sigma}=\{0,1\}, \boldsymbol{\Gamma}=\{0,1,b\}, \mathbf{b}, \mathbf{Q}=\{q_0, q_1, q_F\}, \boldsymbol{q_0}, \boldsymbol{f}, \boldsymbol{F}=\{q_F\})$

donde f:

f	0	1	b
$\rightarrow q_0$	$(q_0, 0, R)$	$(q_1, 1, R)$	$(q_F, 0, S)$
q_1	$(q_1, 0, R)$	$(q_0, 1, R)$	$(q_F, 1, S)$
$*q_F$			

14

Ejemplo de una MT

$MT\_1 = (\boldsymbol{\Sigma}=\{0,1\}, \boldsymbol{\Gamma}=\{0,1,b\}, \mathbf{b}, \mathbf{Q}=\{q_0, q_1, q_F\}, \boldsymbol{q_0}, \boldsymbol{f}, \boldsymbol{F}=\{q_F\})$

donde f:

f	0	1	b
$\rightarrow q_0$	$(q_0, 0, R)$	$(q_1, 1, R)$	$(q_F, 0, S)$
q_1	$(q_1, 0, R)$	$(q_0, 1, R)$	$(q_F, 1, S)$
$*q_F$			

Estado Inicial (→)

Estado Final (*)

También puede representarse como: □

Representación: (Estado al que transita, Símbolo que se escribe en la cinta, Movimiento que realiza el cabezal de L/E)

Desplazamiento:
R -> Derecha
L -> Izquierda
S -> No se mueve

También puede representarse como: +,-,=

15

Ejemplo de una MT

$MT\_1 = (\boldsymbol{\Sigma}=\{0,1\}, \boldsymbol{\Gamma}=\{0,1,b\}, \mathbf{b}, \mathbf{Q}=\{q_0, q_1, q_F\}, \boldsymbol{q_0}, \boldsymbol{f}, \boldsymbol{F}=\{q_F\})$

donde f:

f	0	1	b
$\rightarrow q_0$	$(q_0, 0, R)$	$(q_1, 1, R)$	$(q_F, 0, S)$
q_1	$(q_1, 0, R)$	$(q_0, 1, R)$	$(q_F, 1, S)$
$*q_F$			

16

Ejemplo de una MT

MT_1 = (Σ={0,1}, Γ={0,1,b}, **b**, **Q**={q_0, q_1, q_F}, q_0, **f**, **F**={q_F})

donde f:

f	0	1	b
→q_0	(q_0, 0, R)	(q_1, 1, R)	(q_F, 0, S)
q_1	(q_1, 0, R)	(q_0, 1, R)	(q_F, 1, S)
*q_F			

Sin transiciones → Parada de la MT

Cinta

b	b	1	0	0	1	0	b

Inicio

Estado: q_1

q_F

Ejemplo de una MT

MT_1 = (Σ={0,1}, Γ={0,1,b}, **b**, **Q**={q_0, q_1, q_F}, q_0, **f**, **F**={q_F})

donde f:

f	0	1	b
→q_0	(q_0, 0, R)	(q_1, 1, R)	(q_F, 0, S)
q_1	(q_1, 0, R)	(q_0, 1, R)	(q_F, 1, S)
*q_F			

Cinta

b	b	1	0	0	1	0	b

¿Cómo funciona esta MT?

Al final de la palabra (en el primer *b*), escribe:
0 → Si el número de 1's de la palabra leída es Par
1 → Si el número de 1's de la palabra leída es Impar

Definición de una MT

Diagrama de Estados:

La función de transición también puede describirse en forma de diagrama de estados:

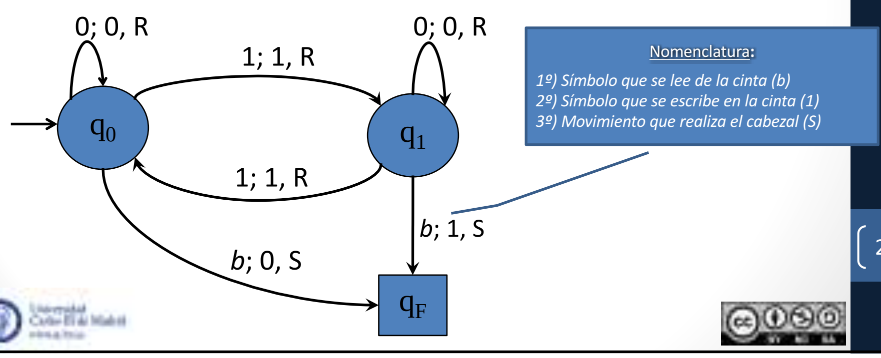
- Los nodos representan estados.
- Los arcos representan transiciones de estados.
- Cada arco es etiquetado con los prerrequisitos y los efectos de cada transición:
  - Símbolo inicial,
  - Símbolo que se escribe,
  - Dirección del movimiento del cabezal.

19

Definición de una MT

Diagrama de Estados - Ejemplo:

f	0	1	b
$\rightarrow q_0$	$(q_0, 0, R)$	$(q_1, 1, R)$	$(q_F, 0, S)$
q_1	$(q_1, 0, R)$	$(q_0, 1, R)$	$(q_F, 1, S)$
$*q_F$			



20

Introducción, Definición y Ejemplos

Tipos de MT

Equivalencia y Variantes de MT

Tipos de MT

MT que actúa como TRANSDUCTOR:

- Modifica el contenido de la cinta realizando cierta función.

MT que actúa como RECONOCEDOR:

- MT capaz de reconocer un lenguaje *L*.
- MT capaz de aceptar un lenguaje *L*.

Tipos de MT

MT que actúa como TRANSDUCTOR:

- Modifica el contenido de la cinta realizando cierta función.

  Ejs: MT que sustituye los dígitos por cero,

  MT que añade un bit de paridad a la entrada,

  MT que duplica el número de 1's que hay en la cinta

  …

- Si la Entrada está bien formada:

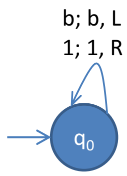
  debe terminar en un Estado Final.

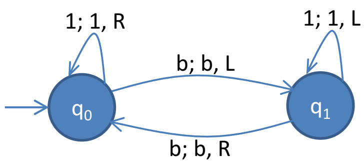
- Si la Entrada No está bien formada:

  debe terminar en un Estado No Final.

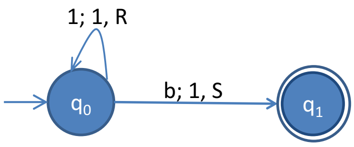
Ejemplos de MT

Diferentes MT que no se detienen ($\Gamma=\{1, b\}$):





MT que calcula $n+1$ considerando el n ($n>=0$) como una sucesión de *1's:*



Tipos de MT

MT que actúa como TRANSDUCTOR:

- Modifica el contenido de la cinta realizando cierta función.

MT que actúa como RECONOCEDOR:

- MT capaz de reconocer un lenguaje *L*.
- MT capaz de aceptar un lenguaje *L*.

25

Tipos de MT

MT que actúa como RECONOCEDOR:

- MT capaz de RECONOCER o ACEPTAR un lenguaje *L*.

- Una MT RECONOCE un lenguaje *L,* si dada una entrada (*w*) en la cinta, la MT SIEMPRE se para, y lo hace en un Estado Final si y sólo si: $w \in L$

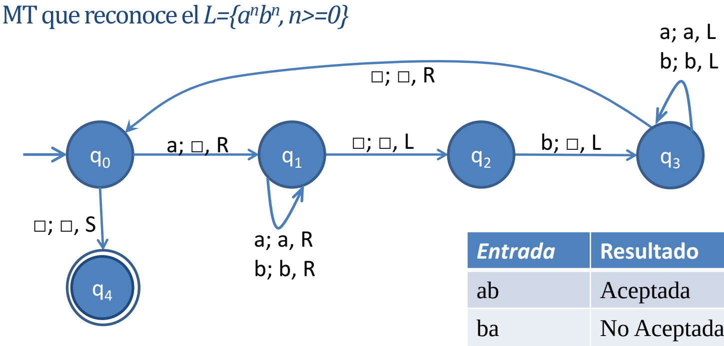
- Una MT ACEPTA un lenguaje *L,* si dada una entrada (*w*) en la cinta, la MT se para en un Estado Final si y sólo si: $w \in L$
  - *Así, en este caso, si* $w \notin L$ *, la MT podría no parar.*

Ejs: MT que reconoce el lenguaje a^*b^**,*
MT que acepta el lenguaje $a^nb^nc^n$
…

26

Ejemplos de MT

MT que reconoce el $L=\{a^n b^n, n>=0\}$



Entrada	Resultado
ab	Aceptada
ba	No Aceptada
aabb	Aceptada
aab	No Aceptada
abb	No Aceptada
aaaabbbb	Aceptada

En este caso, el símbolo especial- espacio en blanco ($b \not\subset \Sigma$) se representa como: □ , porque $b \subset \Gamma$

27

- Introducción, Definición y Ejemplos
- Tipos de MT
- Equivalencia y Variantes de MT

28

Equivalencia de MT

Dos MT son equivalentes si:

Ambas realizan la misma acción sobre TODAS sus entradas. Además, si una MT no se parara para alguna entrada, la otra tampoco podrá pararse.

- Si las MT actúan como Transductor:
  - Para cada entrada posible, los contenidos de la cinta al final del proceso deben ser iguales.
- Si las MT actúan como Reconocedor:
  - Ambas deben Aceptar y/o Reconocer las mismas palabras.

29

Variantes de MT

- Existen numerosas variantes de MT obtenidas al restringir algún aspecto de las mismas.
- Consideremos algunos ejemplos:
  - MT con alfabeto binario ($\Gamma = \{0,1,b\}$).
  - MT limitada por un extremo.
  - MT con restricciones en el movimiento de L/E.

30

La Máquina de Turing

- Breaking the Code: Biography of Alan Turing (Derek Jacobi, BBC, 1996) http://www.youtube.com/watch?v=S23yie-779k
- https://www.youtube.com/watch?v=8fglRhM9pkU
- http://www.youtube.com/watch?v=6k2OUZdA7vQ

31

MT Universal (MTU)

- MT capaz de simular el comportamiento de cualquier MT.
- Una MTU contiene en su cinta:
  1. La descripción de otra MT,
  2. El contenido de la cinta de dicha MT,

  y produce como resultado de su ejecución, el mismo resultado que produciría la MT sobre su cinta.
- Es una MT "programable"

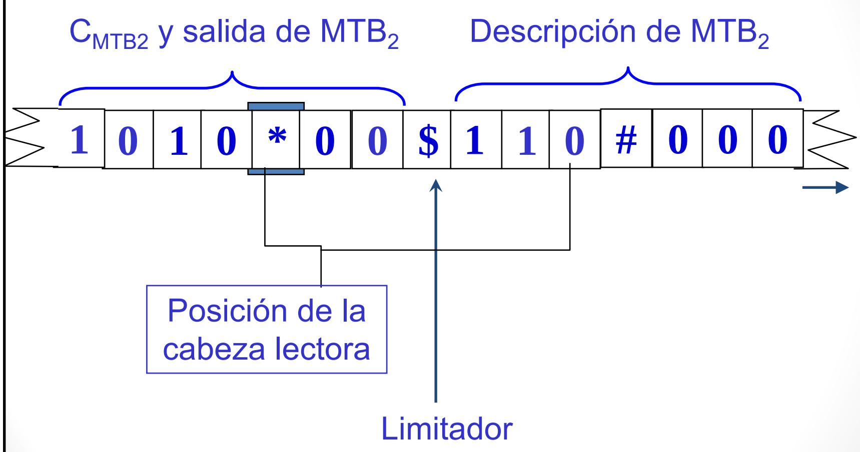
32

MT Universal (I)

- Es una MT "programable"
  - Dependiendo del programa, puede simular a cualquier otra MT.
  - Lee y ejecuta programas almacenados en su cinta
  - El programa de una MTU es una versión codificada de una MT que lleva a cabo la tarea que se desea que ejecute la MTU.
- Para construir una MTU que desempeñe una tarea:
  - Hay que diseñar una MT genérica para esa tarea y
  - codificar dicha MT genérica en la cinta de la MTU.
  - Se tratará de una MT con alfabeto binario y cinta limitada en un sentido.
  - codificar la cadena de entrada.

33

MT Universal



En torno a la figura de Alan Turing

34

MT Universal. Cinta

- Información sobre:
  - función de transición
  - estado inicial
  - posición inicial de la cabeza
  - contenido inicial de la cinta
- Dividida en dos partes por el $:
  - izda del $: CMTB2
  - dcha del $: codificación de MTB2
- Cabeza lectora: ¿tiene que leer en ambos lados del $?
  - Donde está la cabeza: *
  - contenido reemplazado por *: en la posición "l" a la dcha del $.
- A la dcha del $, estructura análoga a MT Transcriptora de Información , con:
  - longitud de la dirección y etiquetas, *"l" = E(log 2 |Q|) + 1*
  - longitud de los registros, *"m" = 2 x l + 1*

35

MT y Computabilidad

- Turing demostró con su MT (y sus extensiones) que todo problema computable (resoluble), lo es en una MTU.
- Complejidad computacional:
  - Se basa en tratar de dar respuesta a la siguiente pregunta:
    - *¿Qué hace a algunos problemas computacionalmente difíciles y a otros sencillos?*
  - Estudia el orden de complejidad de un algoritmo que resuelve un problema *decidible.*
  - Para ello, considera los 2 tipos de recursos requeridos durante el cómputo para resolver un problema:
    - Tiempo: Número de pasos base de ejecución de un algoritmo para resolver un problema.
    - Espacio: Cantidad de memoria utilizada para resolver un problema.

36

Funciones NO computables (I)

- ¿Cuáles son los problemas/funciones no computables?
- Equivalente a los "no-decidibles":
  - **Función de Rado**:
  - **Secuencias aleatorias**
  - **El teorema de Fermat**: $a^n+b^n = c^n$ / a, b, c ≠ 0 y n entero >2
  - **Los primos pares**:

37

Funciones NO computables (II)

- ¿Cuáles son los problemas/funciones no computables?
- Equivalente a los "no-decidibles":
  - **Función de Rado**: El Problema del Castor Afanoso:¿Cuál es el número máximo de 1's que pueden ser escritos por una máquina de Turing de N estados (donde N no incluye el estado final) que termina en parada, y que comienza con una cinta inicialmente en blanco? Este número, que varía en función del número de estados de la máquina, se denota σ (N). Una máquina que produce σ (N) celdas no en blanco se denomina Castor Afanoso.
  - **Secuencias aleatorias**
  - **El teorema de Fermat**
  - **Los primos pares**

38

Funciones NO computables (III)

- Función de Rado:

Num. estados	Num max. 1's impresos	Cota inferior para el valor de σ
3	$\sigma(3)$	6
4	$\sigma(4)$	12
5	$\sigma(5)$	17
6	$\sigma(6)$	35
7	$\sigma(7)$	22.961
8	$\sigma(8)$	3^{92}
9	$\sigma(9)$	$3^{92}+1$
10	$\sigma(10)$	$((a^a)^a)^{\dots a} = a^a$

39

Funciones NO computables (IV)

- ¿Cuáles son los problemas/funciones no computables?
- Diferentes de los "no-decidibles":
  - **Función de Rado**: El Problema del Castor Afanoso:¿Cuál es el número máximo de 1's que pueden ser escritos por una máquina de Turing de *N* estados (donde *N* no incluye el estado final) que termina en parada, y que comienza con una cinta inicialmente en blanco? Este número, que varía en función del número de estados de la máquina, se denota ∑ (N). Una máquina que produce ∑ (N) celdas no en blanco se denomina Castor Afanoso.
  - **Secuencias aleatorias**
  - **El teorema de Fermat**
  - **Los primos pares**:

40

Funciones NO computables (V)

- ¿Cuáles son los problemas/funciones no computables?
- Diferentes de los “no-decidibles”:
  - **Función de Rado**:
  - **Secuencias aleatorias**
  - **El teorema de Fermat**: $a^n+b^n = c^n$ / a, b, c ≠ 0 y n entero >2
  - **Los primos pares**:

Funciones NO computables (VI)

- ¿Cuáles son los problemas/funciones no computables?
- Diferentes de los “no-decidibles”:
  - **Función de Rado**:
  - **Secuencias aleatorias**
  - **El teorema de Fermat**:
  - **Los primos pares**: Existe un número infinito de primos p tales que $p + 2$ también es primo. Ejs: 3 y 5 son primos pares, 11 y 13 ó 29 y 31. según los primos son más grandes la frecuencia de aparición de pares va disminuyendo, pero siguen surgiendo pares de primos gemelos aun entre números de tamaños enormes.

Paradojas

- La **paradoja de Russell** o **paradoja del barbero**, 1901,
  - En un lejano poblado de un antiguo emirato había un barbero llamado As-Samet *diestro en afeitar cabezas y barbas, maestro en escamondar pies y en poner sanguijuelas*. Un día el emir se dio cuenta de la falta de barberos en el emirato, y ordenó que ***los barberos sólo afeitaran a aquellas personas que no pudieran hacerlo por sí mismas***. Cierto día el emir llamó a As-Samet para que lo afeitara y él le contó sus angustias:
    - En mi pueblo soy el único barbero. *No puedo afeitar al barbero de mi pueblo, ¡que soy yo!, ya que si lo hago, entonces puedo afeitarme por mí mismo, por lo tanto ¡no debería afeitarme! Pero, si por el contrario no me afeito, entonces algún barbero debería afeitarme, ¡pero yo soy el único barbero de allí!*
  - El emir pensó que sus pensamientos eran tan profundos, que lo premió con la mano de la más virtuosa de sus hijas. Así, el barbero As-Samet vivió para siempre feliz y barbón.

43

El problema de la Parada (I)

NO existe una Máquina de Turing que pueda decidir si una Máquina de Turing se va a parar: problema de la parada.

- Prueba por reducción al absurdo:
  - Supongamos que si existe esa MT y llamémosla ***A***
  - ***A*** tiene codificado en su cinta (parecido a como ocurre en MTU) una MT (***MTp***) y su cinta (***C***). A la salida hará:

$$A(MTp, C) = \begin{cases} 1 \, si \; MTp \, con \, C \; se \; para \\ 0 \, si \; MTp \, con \, C \; no \; se \; para \end{cases}$$

  - La MT ***A*** cuando escribe un 1, entra en un bucle ∞ (no se para)

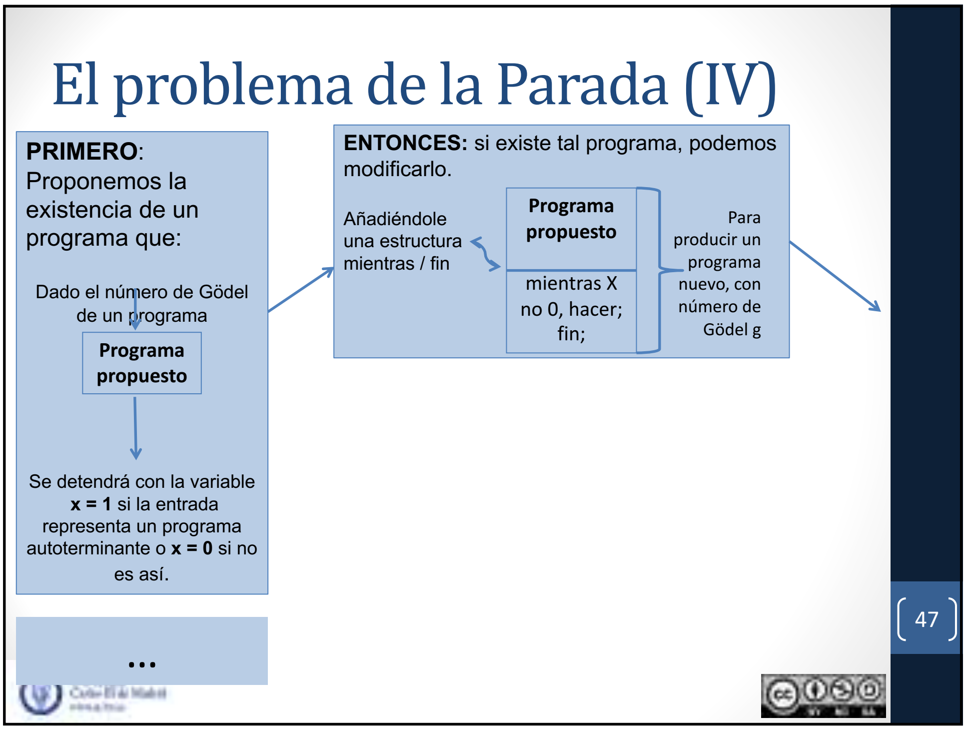
44

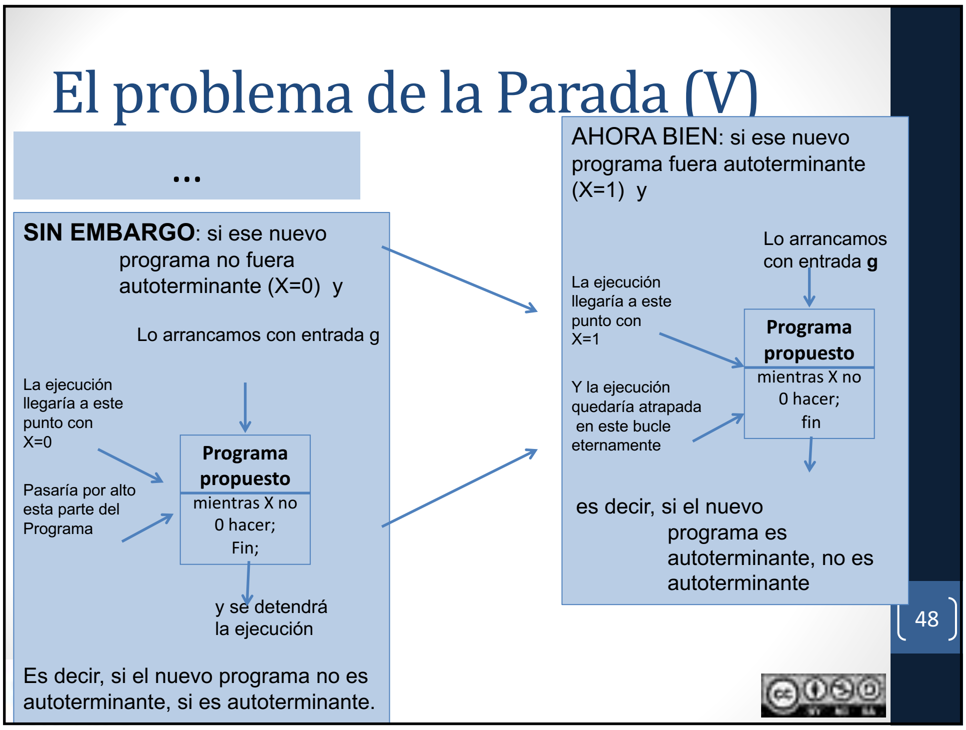
El problema de la Parada (II)

- ***MT_p***: es la p-ésima MT
  - La cinta ***C*** puede ser binaria, de forma que ***C*** se corresponderá con un número binario (***q***).
- Supongamos que escribimos una MT_1 que:
  - Recibe el número en binario *k,*
  - lo duplica y
  - aplica al resultado ***A (MT_k,k)***.
- Dicha máquina será una MT determinada, por ejemplo la *n*-ésima.

El problema de la Parada (III)

- ***¿Qué ocurrirá con A (MT_n,n)?***:
  - Si ***A(MT_n,n)*** se para ➔ MT_n (n) no acaba
    - Si MT_n (n) no se para ➔ A(n,n) acaba
    - Si A (n,n) no se para ➔ MT_n (n) acaba
  - Es decir, si MT_n (n) no se para ➔ MT_n (n) se para, ABSURDO!
- Por otra parte:
  - Si ***A(MT_n,n)*** no se para ➔ MT_n (n) acaba
    - Si MT_n (n) se para ➔ A(n,n) no acaba
    - Si A (n,n) se para ➔ MT_n (n) no acaba
  - Es decir, si MT_n (n) se para ➔ MT_n (n) no se para, ABSURDO!
- Como hemos llegado a un absurdo, la hipótesis de partida es falsa.


El problema de la Parada (IV)
PRIMERO: Proponemos la existencia de un programa que:
Dado el número de Gödel de un programa
Programa propuesto
Se detendrá con la variable x = 1 si la entrada representa un programa autoterminante o x = 0 si no es así.
…
ENTONCES: si existe tal programa, podemos modificarlo.
Añadiéndole una estructura mientras / fin
Programa propuesto
mientras X no 0, hacer; fin;
Para producir un programa nuevo, con número de Gödel g
47


El problema de la Parada (V)
…
SIN EMBARGO: si ese nuevo programa no fuera autoterminante (X=0) y
Lo arrancamos con entrada g
La ejecución llegaría a este punto con X=0
Pasaría por alto esta parte del Programa
Programa propuesto
mientras X no 0 hacer; Fin;
y se detendrá la ejecución
Es decir, si el nuevo programa no es autoterminante, si es autoterminante.
AHORA BIEN: si ese nuevo programa fuera autoterminante (X=1) y
Lo arrancamos con entrada g
La ejecución llegaría a este punto con X=1
Y la ejecución quedaría atrapada en este bucle eternamente
Programa propuesto
mientras X no 0 hacer; fin
es decir, si el nuevo programa es autoterminante, no es autoterminante
48

El problema de la Parada (V)

…

…

En CONSECUENCIA:

La existencia del Programa propuesto

Programa propuesto autoterminante

Conducíría a

la existencia de un nuevo programa

Programa propuesto

mientras X no 0 hacer; fin

que no es ni no autoterminante.

Así que la existencia del programa propuesto es imposible.

49

7. Máquinas de Turing

Grado Ingeniería Informática

Teoría de Autómatas y Lenguajes Formales

Parte VIII

TEMA 8. Complejidad Computacional

Tema 8. Complejidad Computacional.

Tema 8. Complejidad Computacional

Índice:

- Autómatas, Complejidad y Computabilidad.
- Clasificación de los Problemas de Decisión.

Autómatas, Complejidad y Computabilidad

- En la Teoría de la Computación, los tres siguientes áreas:
  - Autómata,
  - Complejidad y
  - Computación

  están relacionados por la siguiente pregunta:
  - *¿Cuáles son las capacidades y limitaciones de los ordenadores?*

- Sin embargo, esta pregunta se interpreta de forma diferente en cada una de las 3 áreas.

Autómatas, Complejidad y Computabilidad

Teoría de Autómatas:

- Se encarga de las definiciones y propiedades de los modelos matemáticos de computación (esenciales en áreas aplicadas de la informática).

- Uno de estos modelos son los Autómatas Finitos, utilizados en:
  - Procesamiento de textos
  - Compiladores
  - Diseño Hardware.
- Otro modelo son las Gramáticas Libres de Contexto, usadas en:
  - Lenguajes de programación
  - Inteligencia Artificial.

Autómatas, Complejidad y Computabilidad

Teoría de la Complejidad:

- Se basa en tratar de dar respuesta a la siguiente pregunta:
  - *¿Qué hace a algunos problemas computacionalmente difíciles y a otros sencillos?*
- Tiene como finalidad la creación de mecanismos y herramientas capaces de describir y analizar la complejidad de un algoritmo y la complejidad intrínseca de un problema.

Autómatas, Complejidad y Computabilidad

Teoría de la Computabilidad:

- Está muy relacionado con la teoría de la Complejidad, ya que introduce varios de los conceptos que esta área utiliza.
- Su finalidad principal es la clasificación de diferentes problemas, así como formalizar el concepto de *computar.*
- Así, estudia qué lenguajes son *decidibles* con diferentes tipos de *"máquinas"* y diferentes modelos formales de computación.

Complejidad Computacional

- Estudia el orden de complejidad de un algoritmo que resuelve un problema *decidible.*
- Para ello, considera los 2 tipos de recursos requeridos durante el cómputo para resolver un problema:
  - Tiempo: Número de pasos base de ejecución de un algoritmo para resolver un problema.
  - Espacio: Cantidad de memoria utilizada para resolver un problema.

Complejidad Computacional

- La complejidad de un algoritmo se expresa como función del tamaño de la entrada del problema, n**.**
- Se refiere al ratio de crecimiento de los recursos con respecto a n:
  - Ratio del Tiempo de ejecución (Temporal): $T(n)$.
  - Ratio del espacio de almacenamiento necesario (Espacial): *S(n).*

Clasificación de Problemas de decisión

- En base a dos criterios:
  - *Teoría de la Computabilidad*
    - *Decidible.*
    - *Parcialmente Decidible (reconocible).*
    - *No Decidible.*
  - *Teoría de la Complejidad Computacional*
    - *Conjuntos de Clase de Complejidad (Clase L, NL, P, P-Completo, NP, NP-Completo, NP-Duro...).*

Un problema de decisión es aquel en el que en el que las respuestas posibles son Si o No

Clasificación de Problemas de decisión

Considerando la Teoría de la Computabilidad un problema de decisión podrá ser:

- Decidible (o resoluble algorítmicamente):
  - Si existe un procedimiento mecánico (MT) que lo resuelva.
  - Además, la MT debe detenerse para cualquier entrada.
- Parcialmente Decidible (Reconocible):
  - Si existe un procedimiento mecánico (MT) que lo resuelva.
  - Además, la MT debe detenerse para aquellas entradas que son una solución correcta al problema.
- No Decidible
  - Si NO es decidible

Clasificación de Problemas de decisión

Considerando la Teoría de la Complejidad Computacional un problema de decisión podrá ser:

- *Conjuntos de Clase de Complejidad (Clase L, NL, P,* P-Completo*, NP, NP-Completo, NP-Duro…).*

En este caso, nos centraremos únicamente en los problemas denominados Clase P, NP y NP-Completo.

Sin embargo, para esta distinción es necesario considerar el modelo teórico de las **Máquinas de Turing**.

Además, debemos distinguir entre:

- **MT Determinista** (Para cada par *(estado, símbolo)*, existe como máximo una transición a otro estado).
- **MT No Determinista** (Existe al menos un par *(estado, símbolo)*, con más de una transición a estados diferentes).

Clasificación de Problemas de decisión

Considerando la Teoría de la Complejidad Computacional un problema de decisión podrá ser:

- Clase P (Polynomial-time)
  - Contiene aquellos problemas de decisión que una **MT Determinista** puede resolver en **tiempo polinómico**.
    - Los problemas de complejidad polinómica son tratables, es decir en la práctica se pueden resolverse en tiempo *razonable.*
    - La mayoría de los problemas *corrientes* (ordenación, búsqueda…) pertenecen a esta clase.

Clasificación de Problemas de decisión

Considerando la Teoría de la Complejidad Computacional un problema de decisión podrá ser:

- Clase NP (Non-Deterministic Polynomial-time)
  - Contiene aquellos problemas de decisión que una **MT No Determinista** puede resolver en **tiempo polinómico**.

Como toda MTD es un caso particular de una MTND:
$P \subseteq NP$

Clase NP
Clase P

Saber si P=NP o P≠NP es todavía un problema abierto en computación teórica!!
Tan importante es demostrar que estas clases son distintas, que es uno de los *problemas* premiados con 1.000.000 $.

http://www.claymath.org/millennium/P_vs_NP/

Clasificación de Problemas de decisión

Considerando la Teoría de la Complejidad Computacional un problema de decisión podrá ser:

- Clase NP-Completo
  - Un problemas de decisión es NP-Completo sii:
    - Es NP
    - Todos los demás problemas de NP se pueden se pueden reducir a él en tiempo polinómico.

Reducir de un problema:
Es una manera de convertir un problema en otro de tal forma que la solución al segundo problema se puede utilizar para resolver el primero.

Tema 8. Complejidad Computacional.

1. RECURSOS

Limpiar y bien formar siempre

1.1. Tema 3

1.1.1. $AFD \rightarrow AFD$ *mínimo*

1. Buscar Q/E_0, que divide los estados en finales y no finales.
2. Hacer $Q/E_1, Q/E_2$... pueden separarse, pero nunca juntarse de nuevo.
3. Cuando se repitan las particiones hemos terminado. Como máximo tendremos que hacer $Q/E_{(n-2)}$ iteraciones.

1.1.2. $AFND \rightarrow AFD$

1. Calcular T^*.
2. Quitar la columna λ y añadir la columna λ^*.
3. Sustituir la columna a, b… por $\lambda^* a \lambda^*$, $\lambda^* b \lambda^*$...
4. El nuevo estado inicial será p λ^*.
5. Transformar los caminos múltiples en estados combinados (finales si alguna de las letras es final) y las transiciones no definidas en transiciones al sumidero.

1.2. Tema 4

1.2.1. *G3 LD* $\rightarrow$ *G3 LI*

1. Quitar el axioma inducido ($A \rightarrow aS$) introduciendo un nuevo símbolo (que haga lo mismo que el axioma, pero no se copia lambda)
2. Construir un grafo dirigido en el que los nodos son los Σ_{NT} y las flechas son los Σ_T.
3. Intercambiar las etiquetas de λ y S, y dar la vuelta a las flechas.
4. Interpretar el grafo

   Nota: las que iban de S van a λ, y de λ no puede salir nada.

1.2.2. Lenguaje vacío (G2)

Generar el árbol de derivación hasta llegar a n (número de estados). Si no genera sentencias y se repiten los Σ_{NT}, es un lenguaje vacío.

1.2.3. Lenguaje infinito (G2)

Construir un grafo cuyos nodos están etiquetados con los Σ_{NT}. Si existen ciclos accesibles desde el axioma, entonces es un lenguaje infinito.

1.2.4. Limpieza y bien-formación de gramáticas

1. Eliminar reglas innecesarias ($A \to A$)
2. Eliminar símbolos inaccesibles (para ello se construye un vector con los símbolos T y NT)

   Ir marcando desde el axioma los que vaya produciendo.
3. Eliminar reglas superfluas con el algoritmo de marcado
   - *a*) Se marcan los $\Sigma_{NT} \to \Sigma_T$ y los $\Sigma_{NT} \to \lambda$
   - *b*) Se marcan los que contengan un Σ_{NT} marcado en la derecha
   - *c*) Se repite hasta que no se pueden marcar más
   - *d*) Se eliminan todas las reglas no marcadas
4. Eliminar los símbolos no generativos, es decir, aquellos que solo aparecen en reglas superfluas.
5. Eliminar las reglas no generativas, las de tipo $A \to \lambda$. Cada vez que aparezca A en la λ. Se admite parte derecha, se añade la posibilidad de que sea $Axioma \to \lambda$, OJO si cuando eliminamos solo quedaba un símbolo ($C \to M$ y eliminamos M), ponemos lambda ($C \to \lambda$) y repetimos el proceso de eliminación (para C).
6. Eliminar las reglas de redenominación, las de tipo $A \to B$. Por cada regla de la forma $B \to x$, se añade $A \to x$

1.2.5. *G2 → FNC*

Se separa el primer símbolo de la derecha del resto, por ejemplo, de $A \to aBb$ sacamos $A \to DE$ y $D \to a$, $E \to Bb \to BC$, $C \to b$ (previo paso debe estar limpia y bien formada)

1.2.6. $G2 \rightarrow FNG$

1. Limpiar y bien formar la gramática.
2. Quitar recursividad a izquierdas si las reglas. λ no se toca en este paso
3. Ordenar el alfabeto Σ_{NT} (A, B) y clasificar las reglas en G2(AB) o G3(BA)
4. Pasar las de G3 a G2. Se hace por sustitución, no sustituir con las reglas que dan λ
    a) Quitar recursividad a izquierdas si apareciese.
5. Pasar de G2 a G1, empezando por la que me deje meter un Σ_T en la cabeza.
6. Si hay un Σ_T que no esté en la cabeza, sustituirlo por un Σ_{NT} que dé ese Σ_T.

1.2.7. Quitar recursividad a izquierdas

Dada una regla de tipo $A \rightarrow A\alpha \mid \beta$ (donde α y β son cualquier cosa...)
Se transforma en:

- $A \rightarrow \beta \mid \beta X$
- $X \rightarrow \alpha X \mid \alpha$

Si tuviéramos varias ($A \rightarrow A\alpha \mid \beta_1 \mid \beta_2$), entonces se transforma en:

- $A \rightarrow \beta \mid \beta X$
- $X \rightarrow \alpha X \mid \alpha$

1.2.8. Paso de G3LD (FNG) → AF y viceversa

Por cada regla $A \rightarrow aB$:

- A y B son estados del autómata y se realiza la transición de A a B con ‘a’.

Si tenemos una regla del tipo $A \rightarrow a$:

- Se realiza la transición de A a un estado final con ‘a’.

Para pasar de AF a G3LD se hace exactamente igual.

1.3. Tema 5

1.3.1. Teoria de síntesis

Nota: $D_{ab}(\alpha) = D_b(D_a(\alpha))$

1. Derivar la expresión respecto de todos los símbolos y todas las que me vayan saliendo.

   Las voy llamando R_x, donde x es un Número

2. Si R_x puede ser λ, se añade una regla, $R_x \rightarrow \lambda$

3. Si $D_y(R_{x_1}) = R_{x_2}$, se añade una regla $R_{x_1} \rightarrow yR_{x_2}$

4. Luego se aplica el paso de G3LD (FNG) a AF para obtener el AF correspondiente

$$
\begin{array}{ll}
D_a(a) = \lambda & \\
D_a(b) = \emptyset & \\
D_a(RS) = D_a(R)S + d(R)D_a(S) & \\
D_a(R + S) = D_a(R) + D_a(S) & \\
D_a(R^*) = D_a(R)R^* & \\
d(a) = \emptyset & \\
d(a^*) = \lambda & \text{Si puede ser } \lambda\text{, es } \lambda\text{, si no } \emptyset \\
d(a^* + a) = \lambda &
\end{array}
$$

1.4. Tema 6

1.4.1. $APF \rightarrow APV$

1. Añadir un estado inicial nuevo con una transición y un nuevo ”chivato” que nos indique cuando se vacía la pila

2. Añadir un estado ”final” que desapile todo lo que pudiera quedar y el chivato.

1.4.2. $APV \rightarrow APF$

1. Añadir un estado inicial nuevo con su chivato

2. Añadir un estado final al que se transita desapilando el chivato

1.4.3. $G2$ (FNG) → APV

1. Tres tipos de reglas:

   a) $A \to aBCD: \ f(q,\ a,\ A) = (q,\ BCD)$

   b) $A \to a: \ f(q,\ a,\ A) = (q,\ \lambda)$

   c) $S \to \lambda: \ f(q,\ \lambda,\ S) = (q,\ \lambda)$

2. El automata tendrá un único estado, q.

1.4.4. APV → $G2$

1. Se pone una regla del tipo $S \to (q_0,\ A_0,\ pqr...)$

2. Se ponen reglas de tipo:

   a) Tipo $1A: \ f(q,\ a,\ B) = (p,\ DEF)$
   El molde será: $(qB\_) \to a(pD\_)(\_E\_)(\_F\_)$

   b) Tipo $1B: \ f(p,\ \lambda,\ B) = (q,\ A)$
   El molde será: $(pB\_) \to X(qA\_)$

   c) Tipo $2A: \ f(p,\ a,\ B) = (q,\ \lambda)$
   El molde será: $(pBq) \to a$

   d) Tipo $2B: \ f(p,\ \lambda,\ B) = (q,\ \lambda)$
   El molde será: $(pBq) \to \lambda$

1.4.5. Equivalencia de EERR

1. $(\alpha+\beta)+\sigma=\alpha+(\beta+\sigma)$
2. $\alpha+\beta=\beta+\alpha$
3. $(\alpha\cdot\beta)\cdot\sigma=\alpha\cdot(\beta\cdot\sigma)$
4. $\alpha\cdot(\beta+\sigma)=(\alpha\cdot\beta)+(\alpha\cdot\sigma)$

   $(\beta+\sigma)\cdot\alpha=(\beta\cdot\alpha)+(\sigma\cdot\alpha)$
5. $\alpha\cdot\lambda=\lambda\cdot\alpha=\alpha$
6. $\alpha+\phi=\phi+\alpha=\alpha$
7. $\lambda^*=\lambda$
8. $\alpha\cdot\phi=\phi\cdot\alpha=\phi$
9. $\phi^*=\lambda$
10. $\alpha^*\cdot\alpha^*=\alpha^*$
11. $\alpha\cdot\alpha^*=\alpha^*\cdot\alpha$
12. $(\alpha^*)^*=\alpha^*$
13. $\alpha^*=\lambda+\alpha+\alpha^2+...+\alpha^n+\alpha^{n+1}\cdot\alpha^*$
14. $\alpha^*=\lambda+\alpha\cdot\alpha^*$
15. $\alpha^*=(\lambda+\alpha)n-1+\alpha n\cdot\alpha^*$
16. Sea f una función, $f:E_{\Sigma^n}\rightarrow E_\Sigma$ se verifica:

    $f(\alpha,\beta,...,\sigma)+(\alpha+\beta+...+\sigma)^*=(\alpha+\beta+...+\sigma)^*$
17. Sea f una función, $f:E_{\Sigma^n}\rightarrow E_\Sigma$ se verifica:

    $(f(\alpha^*,\beta^*,...,\sigma^*))^*=(\alpha+\beta+...+\sigma)^*$
18. $(\alpha^*+\beta^*)^*=(\alpha^*\cdot\beta^*)^*=(\alpha+\beta)^*$
19. $(\alpha\cdot\beta)^*\cdot\alpha=\alpha\cdot(\beta\cdot\alpha)^*$
20. $(\alpha^*\cdot\beta)^*\cdot\alpha^*=(\alpha+\beta)^*$
21. $(\alpha^*\cdot\beta)^*\cdot\alpha^*=\lambda+(\alpha+\beta)^*\cdot\beta$
22. R. Inferencia: $X=Ax+B\rightarrow X=A^*\cdot B$

1.4.6. Analisis

1. Hacer las ecuaciones del Automata Finito

   De X_0 a X_1 con una 'a': $X_0=aX_1$

   De X_0 a X_2 que es final con una 'b': $X_0=bX_2+b$

   Si hay varias transiciones se ponen en la misma ecuación sumándose

   Si el estado final solo va al sumidero o no tiene ramas se le añade λ
2. Utilizar las equivalencias de EERR, empezando por las más lejanas al inicial. Esencialmente se usa la regla de inferencia

$$\begin{array}{ll} X_0=aX_0 & X_0=\emptyset \\ X_1=bX_1+c & X_1=b^*c \\ X_2=c & X_2=c \end{array}$$

1.4.7. Formatos

AFD = (Alfabeto, Q, q_0, f, F) F = Estado finales f=Función transición

AFND = (Alfabeto, Q, q_0, f, F, T) T = Transiciones con λ

G = (Terminales, No Terminales, S, P) S = Axioma P = Transiciones

AP = (Alfabeto cinta, Alfabeto pila, Q, A_0, q_0, f, F) A_0 = Fondo pila

MT = (Alfabeto de entrada, Alfabeto cinta, b, Q, q_0, f, F) b = Símbolos especiales

1.4.8. Jeraquia de Chomsky

Todos aceptan axioma para dar λ

G0 Lenguaje sin restricciones, puede ser cualquier cosa, se caracteriza por reglas compresoras ($aVs \rightarrow d$ OJO también si no es axioma $B \rightarrow \lambda$) y estructura de frases ($AS \rightarrow SA$)

G1 Sensible al contexto, puede ser cualquier cosa, sin reglas compresoras y Contexto ($aS \rightarrow ADC$)

G2 De contexto libre, un símbolo a la izquierda pero cualquier cosa a la derecha. También si hay G3LD y G3LI en la misma gramática.

G3 Gramática regular, son las de la forma $NT \rightarrow T$ y un tipo de las siguientes reglas:

G3LI	$NT \rightarrow NT\ T$
G3LD	$NT \rightarrow T\ NT$