

Grado en Ingeniería Informática
2019-2020

Apuntes

Teoría de Automatas y Lenguajes Formales

Jorge Rodríguez Fraile¹



Esta obra se encuentra sujeta a la licencia Creative Commons
Reconocimiento - No Comercial - Sin Obra Derivada

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

ÍNDICE GENERAL

| | |
|--|----------|
| I Teoría | 3 |
| 1. RECURSOS | 33 |
| 1.1. Tema 3 | 33 |
| 1.1.1. $AFD \rightarrow AFD$ mínimo | 33 |
| 1.1.2. $AFND \rightarrow AFD$ | 33 |
| 1.2. Tema 4 | 33 |
| 1.2.1. $G3\ LD \rightarrow G3\ LI$ | 33 |
| 1.2.2. Lenguaje vacío (G2) | 34 |
| 1.2.3. Lenguaje infinito (G2) | 34 |
| 1.2.4. Limpieza y bien-formación de gramáticas | 34 |
| 1.2.5. $G2 \rightarrow FNC$ | 34 |
| 1.2.6. $G2 \rightarrow FNG$ | 35 |
| 1.2.7. Quitar recursividad a izquierdas | 35 |
| 1.2.8. Paso de G3LD (FNG) \rightarrow AF y viceversa | 35 |
| 1.3. Tema 5 | 36 |
| 1.3.1. Teoría de síntesis | 36 |
| 1.4. Tema 6 | 36 |
| 1.4.1. $APF \rightarrow APV$ | 36 |
| 1.4.2. $APV \rightarrow APF$ | 36 |
| 1.4.3. $G2\ (FNG) \rightarrow APV$ | 37 |
| 1.4.4. $APV \rightarrow G2$ | 37 |
| 1.4.5. Equivalencia de EERR | 38 |
| 1.4.6. Análisis | 38 |
| 1.4.7. Formatos | 39 |
| 1.4.8. Jeraquía de Chomsky | 39 |

Parte I

Teoría

Tema 1: Introducción.

Mirar la primera presentación, sobre la historia y origen. (No entra)

Tema 2: Teoría de Automatas.

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

Computación: Conseguir que una maquina realice un procedimiento.

Automata: Instrumento o aparato que encierra dentro de sí el mecanismo que le imprime determinados movimientos.

Dispositivo abstracto con capacidad de computación.

Teoría de Automatas: Abstracción de cualquier tipo de computador y/o lenguaje de programación.

Tipos de autómatas:

Autómatas finitos (y máquinas secuenciales) AF-MS

Los recordados
entran el resto,
solo salen que existen.

Orientado a reconocer lenguajes. No produce salida, pero las máquinas secuenciales están orientadas para dar una salida.



Autómatas probabilísticos. AP → AFND

Se puede reducir a un Autómatas finito no determinista

Autómatas a pilas AP = AF + memoria pila.

Autómatas finito con una memoria de pila, que sirve para contar.

Células de McCulloch-Pitts Mc-P

En un determinado umbral (paso o avance topel) de activación,

comienza a transmitir información/dar salida. Como el cerebro.

Pero no se pueden separar los datos con más que un plano

Maquina de Turing MT

La maquina con la mayor capacidad computacional, todo lo que se puede escribir lo puede computar.

Está en el nivel más alto de la jerarquía de Chomsky

Automatas celulares. AC

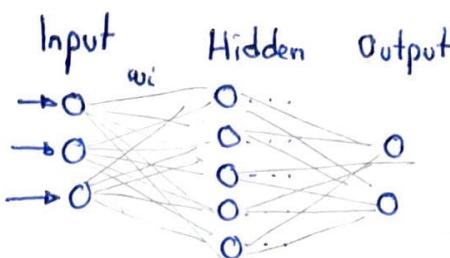
Se basan en las reglas de la vecindad (Ejem: El juego de la vida)
Redes neuronales artificiales RN

Capacidad de aprendizaje.

Percepción multicapa (MLP)

O = Célula de Mc-Pitts

Si supera un cierto peso (w_{ij})
da una salida.



Todo automata puede ser transformado en un algoritmo y a la inversa.
Criterio: Entradas

Suelen ser Discretos: AF (y HS), AP y Maquina de Turing.

Son Discretas, Continuos y/o híbridos: AC y RN Artificiales.

Los AF y AP, gestionan mal los cambios del entorno, no de terminismo.

'No determinismo' AF, AP y MT, para la misma entrada varios estados.

Lenguajes formales:

Problemas: Dado un lenguaje L , generar cadenas que pertenezcan a L

↳ Escribir / Programar.

Dada una cadena x , reconocer si pertenece a L

↳ Diccionario / Explorador sintáctico.

Abstracciones: $L_{finito} = \{a, aaa, aaaaa\}$

$L_{infinito} = \{a^n / n \in \mathbb{N}\}$

↳ Enumeración de términos.

↳ Descripción finita.
(Metalingüística)

Gramática: Descripción metalingüística para desarrollar algoritmos.

$\{A = a \quad \text{Es infinito porque es recursivo, la } A \text{ pasa a ser } aaA.$
 $A = aaA \quad aa\underline{A} + aaaa\underline{A} \rightarrow aaaaag\}$

$L_{inf} = \{a^n / n \text{ par}\}$

Simbolo: Entidad abstracta. Son parte de un alfabeto.

Alfabeto (Σ): Conjunto finito no vacío de simbolos. $\Sigma_2 = \{0, 1\}$ $\Sigma_s = \{\text{ELSE, IF...}\}$

Palabra: Secuencia finita de simbolos del alfabeto. Notación: letras minúsculas del final. $x=001$

Longitud de palabra: Número de simbolos que contiene la palabra. $|x|=3$ $y=1F1F$

Palabra vacía (λ): Palabra cuya longitud es 0. $|\lambda|=0$. Será el elem. neutro en muchas operaciones.

Universo del discurso ($W(\Sigma)$): Conjunto de todas las palabras que se pueden formar con los símbolos del alfabeto Σ . Es un conjunto infinito.
 λ pertenece a todos. $W(\Sigma) = \{\lambda, 0, 1, 00, \dots\}$

Palabra: Operaciones.

Concatenación de palabras: $x \in W(\Sigma) \text{ e } y \in W(\Sigma) \Rightarrow xy = xy$

Prop. asociativa $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

No commutativa

Elem. neutro $\lambda \quad x\lambda = \lambda x = x$

Operación cerrada.

$\overbrace{x}^{\text{cabeza}} \underbrace{y}_{\text{cola}}$

$\overbrace{\text{MiCasa}, \lambda}^{\text{cab}}, \underbrace{\lambda}_{\text{cola}}$

$\overbrace{\lambda}^{\text{cab}}, \underbrace{\text{MiCasa}}_{\text{cola}}$

$\overbrace{\text{Mi}, \text{Casa}}^{\text{cab}}, \underbrace{\lambda}_{\text{cola}}$

Es propia, cuando la opuesta propia no es λ .

Potencia

de una palabra: Reducir la concatenación de la misma palabra.

$$x^0 = \lambda \quad x^1 = x \quad x^2 = x \cdot x = xx$$

Lenguaje (L): Subconjunto del lenguaje universal de Σ , $L \subseteq W(\Sigma)$

L especiales: \emptyset Lenguaje vacío, ningún elemento

$\{\lambda\}$ Lenguaje de la palabra vacía, solo λ

$L \subseteq W(\Sigma)$ Alfabeto, generado por sí mismo.

Concatenación de lenguajes: Sobre el mismo alfabeto.

Las formadas por L_1 , seguido de las del L_2 .

$$L_1 L_2 = L_1 \cdot L_2$$

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

Asociativa

No commutativa

Elem. neutro λ

Operación cerrada.

Unión de lenguajes: Definidos sobre el mismo alfabeto.

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

Conjunto formado indistintamente por palabras de uno u otro lenguaje.

Asociativa

Operación cerrada

Commutativa

Elem. neutro \emptyset

de los dos lenguajes

$$L_1 = \{\text{holo, odio}\}$$

$$L_2 = \{\text{Casa, bota}\}$$

$$L_1 L_2 = L_1 \cdot L_2 = \{\text{holo casa, odio casa, odio bota, holo bota}\}$$

Una de cada

$$L_1 \cup L_2 = \{\text{holo, odio, casa, bota}\}$$

$$L_1 \cap L_2 = \emptyset$$

Tema 3: Autómatas Finitos.

Aplicaciones:

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 un ordenador, sin o con poca preparación previa adicional.

Fases: Análisis y Síntesis.

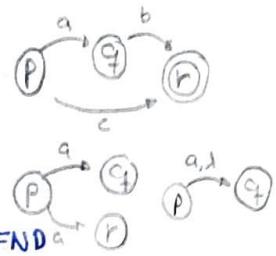
Tipos:

Deterministas: Cada combinación produce un solo estado.

Las transiciones con λ , llevan a sí mismo. $\xrightarrow{P\lambda} P$

No deterministas: Cada combinación produce varios estados.

Son posibles las transiciones con λ , es característico de los AFNDs Con doble círculo.



Representación:

Diagrama de transición:

Nodos etiquetados con los estados. P

Arcos entre nodos etiquetados con símbolos de entrada. $P \xrightarrow{a} Q$

Estado inicial, señalado con \rightarrow . \xrightarrow{P}

Estado final, señalado con * o con doble círculo. P^* o Q



Tabla de transición:

Filas, estados /

Columnas, símbolos de entrada. —

AFD: Se representan con una quintupla $\rightarrow (\Sigma, Q, f, q_0, F)$

Σ Alfabeto de entrada

Q Conjunto de estados, finito y no vacío.

$f: Q \times \Sigma \rightarrow Q$ Función de transición.

$q_0 \in Q$ Estado inicial.

$F \subset Q$ Conjunto de estados finales o de aceptación.

AFD como reconocedor de Lenguajes:

Cuando transita desde q_0 (inicial) a un estado final en varios movimientos, se ha producido el Reconocimiento o Aceptación de la cadena de entrada.

Cuando no es capaz de alcanzar un estado final, el AF no reconoce la cadena y ésta no pertenece al lenguaje reconocido por el AF

Conceptos básicos:

Configuración: es un par ordenado de la forma (q, w) , donde:

q estado actual del AFD

w Cadena que le queda por leer. $w \in \Sigma^*$

Configuración inicial: (q_0, t)

q_0 estado inicial.

t cadena de entrada a reconocer por el AFD

Configuración final: (q_f, λ)

q_f estado final

λ la cadena de entrada ha sido leída completamente.

Movimiento: es el tránsito entre dos configuraciones.

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

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

f solo contiene palabras de longitud 1, no se usa conf.

f' contiene palabras de longitud mayor que 1, se usa λ conf' $x \in \Sigma^*$

$$f'(p, \lambda) = p \quad f'(p, aba) = r \quad f'(p, aa) = f'(f(p, a), a) = f'(q, a) = r$$

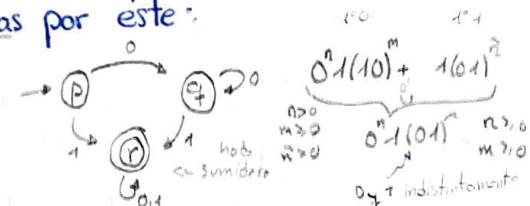
Lenguaje asociado a un AFD

Conjunto de todas las palabras aceptadas por éste:

$$L = \{x / x \in \Sigma^* \text{ and } f'(q_0, x) \in F\}$$

Si $F = \{\} = \emptyset \Rightarrow L = \emptyset$

Si $F = Q \Rightarrow L = \Sigma^*$



Estados accesibles: $p \in Q$ es accesible desde $q \in Q$ si $\exists x \in \Sigma^* / f'(q, x) = p$.

En caso contrario es inaccesible.

Teoremas: $|Q| = n, \forall p, q \in Q$ p es accesible desde $q \xrightarrow{\text{si } f'(q, x) = p} f(x \in \Sigma^*, |x| < n) / f'(q, x) = p$

Solo AFD No palabras más largas que el nº de estados.

$|Q| = n, L_{AFD} \neq \emptyset \xrightarrow{\text{solo}} AFD \text{ acepta al menos una palabra } x \in \Sigma^*, |x| < n$.

Autómatas conexos: Si todos los estados de Q son accesibles desde q_0

Para hacerlo conexo eliminamos los inaccesibles desde q_0

Autómatas equivalentes: Automata que reconoce el mismo lenguaje que otro.

Es posible tener varios automatas que reconozcan el mismo lenguaje, y es

posible obtener uno con el menor número de estados.

Equivalencia: Dos estados son equivalentes si al recibir la misma cadena llegan a estados finales ambos.

que lleva a uno y vuelven al otro.

Teoremas:

Equivalencia de estados: $p E q$, donde $p, q \in Q$, si $\forall x \in \Sigma^*$ se verifica:

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

Equivalencia de orden (o longitud) "n": Siendo $n = |Q| - 2$

$p E_n q$, donde $p, q \in Q$, si $\forall x \in \Sigma^*/|x| \leq n$ verifica: $f'(p, x) \in F \Leftrightarrow f'(q, x) \in F$

E_0 : Todos los estados finales lo son entre ellos, $f(p, \lambda) \in F$ y $f(q, \lambda) \in F$; $p \in F$ y $q \in F$
 $E_0 = \{F, \bar{F}\}$ y los que no son, son equivalentes entre sí. $\begin{array}{c} \textcircled{a} \rightarrow \textcircled{b} \rightarrow \textcircled{d} \\ \textcircled{c} \leftarrow \textcircled{b} \end{array}$ $E_0 = \{\{c, b\}, \{a, d\}\} = \{\{F\}, \{\bar{F}\}\}$

$$p E q \Leftrightarrow p E_{n-2} q, \text{ donde } n = |Q| - 1$$

Lema: $p E q \Rightarrow p E_n q \quad \forall n, p, q \in Q$

Lema: $p E_n q \Rightarrow p E_k q \quad \forall n > k$

Lema: $p E_{n+1} q \Leftrightarrow (p E_n q \text{ and } f(p, a) E_n f(q, a)) \forall a \in \Sigma$

Ej: x palabra, $|x| \leq 1, (x \in \Sigma)$ verifica $p E_1 q, \forall p, q \in Q$, si $\forall x \in \Sigma^*/|x| \leq 1$

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

"E" es una Relación de equivalencia.

" Q/E " es una partición de Q

$$Q/E = \{C_1, C_2, \dots, C_i\}, \text{ donde } C_i \cap C_j = \emptyset$$

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

Propiedades (Lemas):

Lema: Si $Q/E_n = Q/E_{n+1} \Rightarrow Q/E_n = Q/E_{n+i} \quad \forall i = 0, 1, \dots$

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: $n = |Q| - 1 \Rightarrow Q/E_{n-2} = Q/E_{n-1}$

$p E_{n+1} q \Leftrightarrow (p E_n q \text{ and } f(p, a) E_n f(q, a) \forall a \in \Sigma)$ Si tengo el $p E_n q$, con una mala pata sobre el siguiente

Comparamos Q/E_0 y Q/E_1 , si son iguales hemos llegado a Q/E , si no vamos a por Q/E_2 .

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

Si hay varias iguales, lo hacemos t

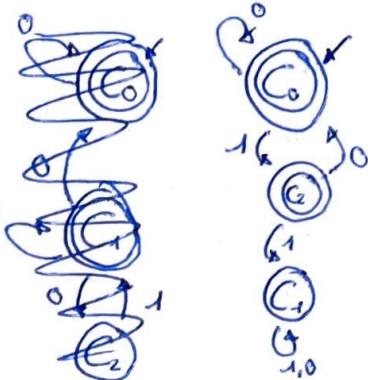
En cuanto se obtienen dos particiones consecutivas $Q/E_k = Q/E_{k+1}$, se para.

Para llegar a Q/E , se empieza por $Q/E_0, Q/E_1$, etc.

Hay que obtener Q/E_{n-2} en el peor de los casos.

Permite extender la equivalencia de orden n desde E_0 y E_1 .

| | 0 | 1 | Q/E_0 | Q/E_1 | Q/E_2 | Q/E_3 |
|------------------|----|---|---------|---------------------|-----------|---------|
| $\rightarrow *A$ | B | C | C_0 | $C_0 C_0 + C_0 C_1$ | $C_3 C_2$ | C_0 |
| $*B$ | B | D | C_0 | $C_0 C_0 + C_0 C_2$ | $C_3 C_2$ | |
| $*C$ | B | E | C_0 | $C_0 C_1 + C_0 C_1$ | $C_3 C_1$ | C_2 |
| $*D$ | B | E | C_0 | $C_0 C_1 + C_0 C_1$ | $C_3 C_1$ | $-C_1$ |
| E | EE | | C_1 | $C_1 C_1 + C_1 C_1$ | $C_3 C_1$ | |



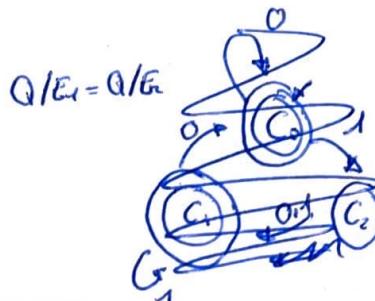
$$C_0 = C_0 C_0 \quad C_1 = C_1 C_1 \quad C_2 = C_0 C_1 \quad \cancel{C_3 = C_0 C_2} \quad C_4 = C_3 C_2 \quad C_5 = C_3 C_1$$

$$Q/E_0 = \{\{A, B, C, D\}, \{E\}\}$$

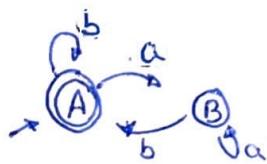
$$Q/E_1 = \{\{A, B\}, \{C, D\}, \{E\}\}$$

$$Q/E_2 = \{\{A, B\}, \{C, D\}, \{E\}\}$$

$\rightarrow *$ * NOTA



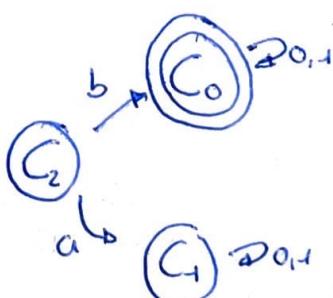
| | a | b | Q/E_0 | a | b |
|-----------------|---|---|---------|-------------|-------|
| $\rightarrow P$ | q | p | C_1 | $C_0 C_1 +$ | |
| $*q$ | r | p | C_0 | $C_0 C_1$ | C_0 |
| $*r$ | q | p | C_0 | $C_0 C_1$ | A |
| $*w$ | w | v | C_0 | $C_0 C_1$ | B |
| $\rightarrow V$ | w | v | C_1 | $C_0 C_1$ | |



$$Q/E_0 = \{\{p, v\}, \{q, r, w\}\}$$

$$Q/E_1 = \{\{p, v\}, \{q, r, w\}\}$$

| | a | b | Q/E_0 | a | b |
|-----------------|---|---|---------|-----------|-----------|
| $*P$ | p | q | C_0 | $C_0 C_0$ | $C_0 C_0$ |
| $*q$ | r | p | C_0 | $C_0 C_0$ | $C_0 C_0$ |
| $*r$ | r | r | C_0 | $C_0 C_0$ | $C_0 C_0$ |
| $\rightarrow S$ | t | p | C_1 | $C_1 C_0$ | $C_1 C_0$ |
| t | b | u | C_1 | $C_1 C_1$ | $C_1 C_1$ |
| u | t | v | C_1 | $C_1 C_1$ | $C_1 C_1$ |
| V | v | u | C_1 | $C_1 C_1$ | $C_1 C_1$ |



$$Q/E_0 = \{\{p, q, r\}, \{s, t, u, v\}\}$$

$$Q/E_1 = \{\{p, q, r\}, \{s\}, \{t, u, v\}\}$$

$$Q/E_2 = \{\{p, q, r\}, \{s\}, \{t, u, v\}\}$$

2^o Llenar tabla con estados y transiciones.
 3^o Miro si $(s_i, a) = t$
 ten Q/E_1 esta en $C_1 \Rightarrow$ Pongo C_1

4^o Llenar Q/E_0 Finales y Nfinales (No necesaria)

3^o Q/E_1 Segun la transicion, mirando Q/E_0 ponemos C_0/C_1

4^o Ver grupos y nombrarlos $C_0/C_1/C_2$

5^o Q/E_2 Mirar transiciones y completar con el grupo al que pertenezcan en Q/E_1

| | 0 | 1 | Q/E_0 | Q/E_1 | Q/E_2 |
|-----------------|---|---|---------|-----------|---------|
| $\rightarrow A$ | B | C | C_0 | $C_0 C_0$ | |
| $*B$ | B | D | C_0 | $C_0 C_0$ | x |
| $*C$ | B | E | C_0 | $C_0 C_1$ | x |
| $*D$ | B | E | C_0 | $C_0 C_1$ | |
| E | E | C | C_1 | $C_1 C_1$ | $-x$ |

$$Q/E_0 = \left\{ \left\{ C_0 = A, B, C, D \right\}, \left\{ C_1 = E \right\} \right\}$$

$$Q/E_1 = \left\{ \left\{ C_0 = A, B \right\}, \left\{ C_1 = E \right\}, \left\{ C_2 = C, D \right\} \right\}$$

$Q/E_0 \neq Q/E_1$

| | 1 | Q/E_0 | Q/E_1 | Q/E_2 | Q/E_3 | Q/E_4 |
|------|---|---------|---------|---------|-------------|-------------------|
| $*F$ | P | C_2 | C_3 | C_3 | $C_3 \dots$ | $C_3 \quad C_4$ |
| P | a | C_1 | C_1 | C_1 | | $(C_1) \quad C_2$ |
| a | b | C_1 | C_1 | C_1 | | C_2 |
| b | c | C_1 | C_1 | C_1 | C_2 | C_3 |
| c | d | C_1 | C_1 | C_2 | C_5 | C_5 |
| d | F | C_1 | C_2 | C_4 | C_4 | C_4 |

$$Q/E_0 = \left\{ \left\{ C_1 = P, a, b, c, d \right\}, \left\{ C_2 = F \right\} \right\}$$

$$Q/E_1 = \left\{ \left\{ C_1 = P, a, b, c \right\}, \left\{ C_3 = d \right\}, \left\{ C_2 = F \right\} \right\}$$

$$Q/E_2 = \left\{ \left\{ C_1 = P, a, b, d \right\}, \left\{ C_4 = C \right\}, \left\{ C_2 = F \right\}, \left\{ C_3 = d \right\} \right\}$$

$$Q/E_3 = \left\{ \left\{ C_1 = P \right\}, \left\{ C_6 = a \right\}, \left\{ C_5 = b \right\}, \left\{ C_4 = c \right\}, \left\{ C_5 = F \right\}, \left\{ C_3 = d \right\} \right\}$$

| | 0 | 1 | E_0 | E_1 | E_2 | E_3 | |
|-----------------|---|---|-------|-----------|-----------|-----------|--------|
| $*F$ | F | F | C_0 | $C_0 C_0$ | $C_0 C_0$ | $C_0 C_0$ | C_0 |
| $*E$ | E | E | C_0 | $C_0 C_0$ | $C_0 C_0$ | $C_0 C_0$ | |
| $\rightarrow A$ | B | C | C_1 | $C_1 C_1$ | $C_2 C_2$ | $C_2 C_2$ | $-C_3$ |
| B | E | D | C_1 | $C_0 C_1$ | $C_0 C_1$ | $C_0 C_1$ | C_2 |
| C | F | D | C_1 | $C_0 C_1$ | $C_0 C_1$ | $C_0 C_1$ | |
| D | D | D | C_1 | $C_1 C_1$ | $C_1 C_1$ | $C_1 C_1$ | $-C_1$ |

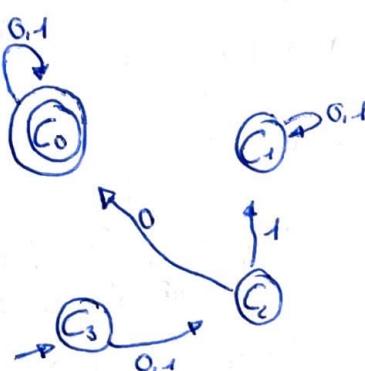
| | Q/B_0 | Q/B_1 | Q/B_2 | Q/B_3 |
|-----------------|-------------|-------------|-------------|-------------|
| $C_0 = F E$ | $C_0 = F E$ | $C_0 = F E$ | $C_0 = F E$ | $C_0 = F E$ |
| $C_1 = A B C D$ | $C_1 = A D$ | $C_1 = D$ | $C_1 = D$ | $C_1 = D$ |
| | $C_2 = B C$ |
| | | $C_3 = A$ | $C_3 = A$ | $C_3 = A$ |

$$C_0 = C_0 C_0$$

$$C_1 = C_1 C_1$$

$$C_2 = C_0 C_1$$

$$C_3 = C_2 C_2$$



Teorema:

$p \in q \Leftrightarrow p \in_{n-2} q$ donde $|Q|=n>1$

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

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

Algoritmo formal para obtener Q/E:

1º Eliminamos los estados inaccesibles.

2º $Q/E_0 = \{F, \bar{F}\}$ 1º División en función de si es final o no.

Mirando Q/E_0 3º Q/E_i partiendo de $Q/E_0 = \{C_1, C_0, \dots\}$ se construye Q/E_{i+1} : p y q están en la misma

Mirando Q/E_1 los conjuntos, ^{y poner las variables} nombrarlos Clase si: $p, q \in C_k \in Q/E_1$ $\forall a \in \Sigma \Rightarrow f(p, a) \in C_m \in Q/E_1$

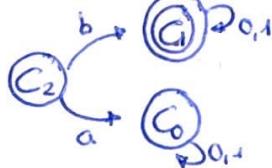
4º Si $Q/E_i = Q/E_{i+1}$ entonces $Q/E_i = Q/E$

Sino, repetir el paso 3 partiendo de Q/E_{i+1}

Ejem:

| | | Q/E_0 | | Q/E_1 | | Q/E_2 | | Q/E_3 | |
|---------------|---|---------|---|----------------|----------------|----------------|----------------|----------------|----------------|
| | | a | b | C ₁ | C ₀ |
| * | P | P | q | C ₁ | C ₀ |
| * | q | r | p | C ₁ | C ₀ |
| * | r | r | r | C ₁ | C ₀ |
| \rightarrow | s | t | p | C ₀ |
| | t | t | u | C ₀ |
| | u | t | v | C ₀ |
| | v | v | u | C ₀ |

$$Q/E_1 = Q/E_2 \Rightarrow Q/E$$



AF Determinista mínimo equivalente

$$b(a+b)^n / n > 0$$

Automatas Equivalentes:

• Estados equivalentes en AFD's distintos:

Sean 2 ADF'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 ($p \in q$) si se verifica que

$$f''(p, x) \in F \Leftrightarrow f''(q, x) \in F' \quad \forall x \in \Sigma^*$$

• Estados equivalentes en AFD's distintos:

Dos ADF's son equivalentes si reconocen el mismo lenguaje, es decir:

$$\text{Si } f(q_0, x) \in F \Leftrightarrow f'(q'_0, x) \in F' \quad \forall x \in \Sigma^*$$

Dos ADF's son equivalentes si lo son sus estados iniciales: $q_0 \in q'_0$

Métodos:

Hacer los mínimos de los dos y ver si son isomorfos (que solo varía la etiqueta de estados)

Hacer el AF sumarlos y veo si $q_0 \in q_0'$

Suma directa de 2 AFD's

$$A_1 = (\Sigma, Q_1, f_1, q_{01}, F_1) \quad \left. \begin{array}{l} \text{Donde } Q_1 \cap Q_2 = \emptyset \text{ y } \Sigma = \Sigma' \end{array} \right\}$$

$$A_2 = (\Sigma', Q_2, f_2, q_{02}, F_2) \quad \left. \begin{array}{l} A = A_1 + A_2 = (\Sigma, Q_1 \cup Q_2, f, q_0, F_1 \cup F_2), \text{ donde} \\ q_0 \text{ es el estado inicial de uno de los AF's} \end{array} \right\}$$

$$f: f(p, a) = f_1(p, a) \text{ si } p \in Q_1$$

$$f(p, a) = f_2(p, a) \text{ si } p \in Q_2$$

Teatema: Sean $A_1, A_2 / Q_1 \cap Q_2 = \emptyset, |Q_1| = n_1, |Q_2| = n_2$

$A_1 \equiv A_2$ si $q_{01} \in q_{02}$ en $A = A_1 + A_2$. Si A_1 y A_2 aceptan las mismas palabras $x / |x| \leq n_1 + n_2 - 2$

Algoritmos para comprobar la equivalencia de AFD's:

1. Se hace la suma directa de los dos AFD's

2. Se hace QIE del AFD suma.

3. Si los dos estados iniciales están en la misma clase de equivalencia de QIE \Rightarrow los 2 AFD's son equivalentes.

Autómatas Finitos No Deterministas.

Definición:

1. AFND = (Σ, Q, f, q_0, F) donde $f: Q \times (\Sigma \cup \lambda) \rightarrow P(Q)$ es No Determinista.

2. AFND = $(\Sigma, Q, f, q_0, F, T)$ donde $f: Q \times \Sigma \rightarrow P(Q)$ conjunto de las partes de Q

T : Relación definida sobre pares de elementos de Q .

Definición formal de la transición λ . $pTq = (p, q) \in T; f(p, \lambda) = q$

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

$$f(p, a) = \{q\} \quad f(p, b) = \{\}$$

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

$$f(r, a) = \{\} \quad f(r, b) = \{p, s\}$$

$$f(s, a) = \{\} \quad f(s, b) = \{\}$$

| | a b λ | | |
|---|--------------------|---------|------|
| | p | q | r |
| q | | p, r, s | p, r |
| r | | | p, s |
| s | | | r |

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 la palabra.

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

1. $f''(q, \lambda) = \{p / qT^* p \forall q \in Q\}$ donde se cumple que $q \in f'(q, \lambda)$

2. sea $x = a_1 a_2 a_3 \dots a_n; n > 0$

$f''(q, x) = \{p / p \text{ es accesible desde } q \text{ por medio de la palabra } \lambda^* a_1 \lambda^* a_2 \lambda^* a_3 \dots \lambda^* a_n \lambda^*\}$

$$\forall q \in Q$$

Calculo de T^*

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

$\lambda \rightarrow \lambda^*$

- Para calcular f'' es necesario extender las transiciones con una λ o λ^* , es decir calcular T^* del AFND = $(\Sigma, Q, f, q_0, F, T)$

- Para ello existe el método formal de las matrices booleanas, o el método de la matriz de pares (estado, estado)

Método de la matriz de pares de estados.

Matriz con tantas filas como estados.

$$T^* = \begin{pmatrix} (p,p) & \text{el propio} \\ (q,q) & \text{de } T \\ (r,r) & f(q,r) = (q,r) \\ (s,s) & f(r,s) = (r,s) \\ (s,s) & f(s,s) = (s,s) \end{pmatrix}$$

En la 1^a columna, el par correspondiente al estado en cuestión (p, p), cada estado es accesible desde sí mismo.

En las columnas siguientes se añaden las transiciones λ definidas en el AFND, considerando si el hecho de añadirlo permite extender alguna transición más.

Cuando no se pueden añadir ningún par más, se habrá terminado T^* .

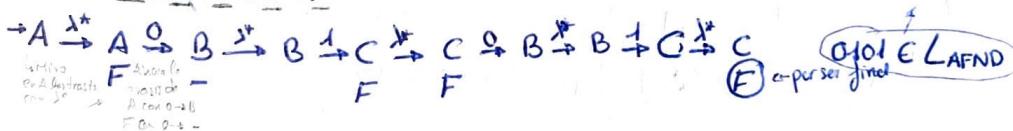
Se extiende la tabla anterior para contener T^* , insertando λ^* (en那些 que van desde $p \rightarrow q, q \rightarrow r, s \rightarrow r$)

| $\rightarrow *P$ | a | b | λ^* |
|------------------|-----------|-----------|---------------|
| q | P | $\sim P$ | $\sim (P, P)$ |
| r | R, R, S | R, R, S | $\sim (R, R)$ |
| s | S | S | $\sim (S, S)$ |

$$T^* = \begin{pmatrix} (A,A) & (A,F) \\ (B,B) & (C,C) \\ (C,C) & (F,F) \end{pmatrix}$$

| $\rightarrow A$ | 0 | 1 | λ | λ^* |
|-----------------|-----|--------|-----------|-----------------|
| B | - | F | F, A | $\sim (A, A)$ |
| C | - | C, F | - | $\sim (C, C)$ |
| F | - | - | - | $F \sim (F, F)$ |

$010 \rightarrow \lambda^* 0 \times 1 \lambda^* 0 \times 1 \lambda^*$



AFND \rightarrow AFD (Ejemplo TALF P4)

26/9/19

1º Calcular λ^* (con T^*)

2º Calcular λ^* símbolo λ^* vsímbolo $\in \Sigma$

3º Transformar los caminos múltiples en estados combinados.

$p, q, r \rightarrow \{pqr\}$ estado combinado. Con las características de p, q, r y alguno es

final $\{pqr\} \in F$

La unión de los estados que afecta
Hay que poner todos

4º Vamos construyendo el automata finito determinista conexo.

Mirando λ^*, λ^* símbolo λ^*, \dots Queremos a, b, \dots, λ^* , solo λ^* símbolo λ^*

5º Las transiciones no definidas van al sumidero.

Si originalmente no hay transiciones λ se empieza en el 3º

| $\rightarrow P$ | a | b | c | $\lambda^* a \lambda^*$ | $\lambda^* b \lambda^*$ | $\lambda^* c \lambda^*$ |
|-----------------|-----|------|-----|-------------------------|-------------------------|-------------------------|
| q | P | q | - | P, q, r | P, q, r | P, q, r |
| r | q | pr | - | pqr | pqr | pqr |
| s | - | - | s | pqr | pqr | s |
| $\rightarrow S$ | s | - | - | s | - | - |

Cuando miro $\{pqr\} \rightarrow \{pqr\}$ miro los 3 valores

oyes si pqr y otro pqr $\Rightarrow \{pqr\}$

| $\rightarrow P$ | a | b | c | $\lambda^* a \lambda^*$ | $\lambda^* b \lambda^*$ | $\lambda^* c \lambda^*$ |
|-----------------|-------------|-------------|-------------|-------------------------|-------------------------|-------------------------|
| $\rightarrow P$ | $\{pqr\}$ | $\{pqr\}$ | S | $\{pqr\}$ | $\{pqr\}$ | S |
| $\rightarrow P$ | $\{pqr\}$ | $\{pqr\}$ | S | $\{pqr\}$ | $\{pqr\}$ | S |
| $\rightarrow S$ | \emptyset | \emptyset | \emptyset | \emptyset | \emptyset | \emptyset |
| $\rightarrow S$ | \emptyset | \emptyset | \emptyset | \emptyset | \emptyset | \emptyset |

Y cogido todos los distintos y
que lo cumple

Tabla

1º Cogemos el est. inicial
2º Completamos orbic mirando
 $\lambda^* a = a, \dots$

3º Cogemos uno de los estat
y lo anidamos el signal

4º Completamos mirando pqr ,
para $a \rightarrow pqr \Rightarrow \{pqr\}$
 $b \rightarrow \emptyset$ $c \rightarrow \emptyset$

Equivaleentes
Miro pqr a la vez
en la tabla anterior
y hago unión de
 a, b, c y final.

5º Cogemos el siguiente
estato y lo anidamos,
así con todos

Tema 4: Lenguajes y Gramáticas formales.

Operaciones con palabras:

Concatenación de palabras

Potencia

Reflexión: Sea la palabra $x = a_1 \cdot a_2 \cdot a_3 \cdots a_n$, se denomina palabra reflexa de x , $\bar{x} = a_n \cdot a_{n-1} \cdots a_2 \cdot a_1$ formada por los mismos símbolos en distinto orden $|x'| = |x|$

Operaciones con Lenguajes:

Unión

Concatenación

Binoide libre: El alfabeto Σ es un conjunto de generadores para el conjunto $L \Rightarrow L$ es el Binoide LIBRE (operaciones \cup y \cdot) generado por Σ . Ya que la concatenación y unión son monoide, con ellos y Σ se puede formar cualquier lenguaje excepto \emptyset y $\{\lambda\}$.

Potencia

Clausura o Cierre positivo (L^+): 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 , excepto si $\lambda \in L$.

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i = W(\Sigma) - \{\lambda\} \quad \text{Siendo } \Sigma \text{ un lenguaje.}$$

Iteración, clausura o cierre (L^*): Es el lenguaje obtenido uniendo el lenguaje L con todas sus potencias positivas. $L^* = \bigcup_{i=0}^{\infty} L^i$

Toda clausura contiene λ . (*) es el operador unario de kleene)

$$L^* = L^+ \cup \{\lambda\} \quad L^+ = L^* \cdot L = L \cdot L^* \quad \Sigma^* = W(\Sigma)$$

Reflexión (L^{-1}): Lenguaje formado por todas las palabras reflejas de L . $L^{-1} = \{\bar{x} \mid x \in L\}$

Reglas de derivación:

Producciones: Sea Σ un alfabeto, se llama producción a un par ordenado (x, y) donde $x, y \in \Sigma^{* \cup \{W(\Sigma)\}}$ $x \in \Sigma^*$ parte izquierda $y \in \Sigma^*$ parte derecha.

Se representa $x ::= y$

Derivación directa: $x \rightarrow y$

ME ::= BA
 CA \rightarrow C A P A L L O
 Produce directamente
 PERA \rightarrow C A R A
 PE ::= CA

si x es un conjunto de producciones

Sea Σ un alfabeto, $x ::= y$ una producción sobre Σ y, v y w dos palabras sobre Σ

Se dice "w es derivación directa de v" o "v produce directamente w" $v \rightarrow w$

Si \exists dos palabras $z, u \in \Sigma^*$ tales que $v = zxu$ y $w = zu$ y se cumple $(x ::= y) \in P$

Si $x ::= y$ es una producción sobre $\Sigma \Rightarrow x \rightarrow y$

Derivación: $x^+ \rightarrow w$

Sea Σ un alfabeto, P un conjunto de producciones sobre Σ y, v y w dos palabras sobre Σ .

Se dice que "w es derivación de v" o "v produce w" $v^+ \rightarrow w$

si \exists una secuencia finita de palabras, $u_0, u_1, \dots, u_n (n > 0) \in \Sigma^*$ tales que $v = u_0$

$u_0 \rightarrow u_1$
 $u_1 \rightarrow u_2$
 \vdots
 $u_{n-1} \rightarrow u_n$
 $w = u_n$

Derivación de longitud n

Relación de Thue

Sea Σ un alfabeto, P un conjunto de producciones y, v y w dos palabras sobre Σ .

Se dice que existe una relación de Thue entre v y w y se representa por

$v^* \rightarrow w$ si se verifica que:

$v^+ \rightarrow w$ o } o \exists una derivación de longitud n o
 $v = w$ } Son iguales.

Propiedades: Reflexiva

Transitiva

Simétrica

(En general no se cumple)

Gramáticas Formales.

¿Qué es 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.

Es un ente formal para especificar de manera finita el conjunto de cadenas de símbolos que constituyen un lenguaje.

Ej: $L_0 = 0^n / n$ par

L_0 es infinito

$\lambda, 00, 0000$

$\begin{cases} A = \lambda & \lambda \\ A = 00A & 00A \xrightarrow{\text{odd}} 0000A \xrightarrow{\text{odd}} 0000 \end{cases}$

Como se define una gramática:

Minúsculas $\rightarrow \Sigma_T$: Alfabeto de símbolos terminales

$\Sigma_T \cup \Sigma_N = \Sigma$

Mayúsculas $\rightarrow \Sigma_N$: Alfabeto de símbolos no terminales

$\Sigma_T \cap \Sigma_N = \emptyset$

S : Axioma

P : Conjuntos de reglas de producción.

Se empieza por el axioma y se hace una sustitución por paso.

Forma sentencial:

Sea 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:

$S^* \rightarrow x$) Es decir, que existe una relación de Thue entre el axioma y x .

$S = S$ } Si $x \in \Sigma_T^*$ se dice que x es una sentencia o instrucción del lenguaje descrito por G .

Sentencia, palabra, cadena o tira: Lenguaje asociado a una gramática.

$L = \{x / S^* \rightarrow x / x \in \Sigma_T\}$ Conjunto de todas las sentencias

Recursividad:

Una G se llama recursiva en U , $U \in \Sigma_N$, si se cumple:

$U^+ \rightarrow xUy$

Si $x = \lambda (U^+ \rightarrow Uy)$ se dice que G es recursiva a izquierdas.

Si $y = \lambda (U^+ \rightarrow xU)$ se dice que G es recursiva a derechas.

Una regla de producción es recursiva si tiene la forma: $U ::= xUy$

Si una gramática es recursiva, el lenguaje es infinito.

Jerarquía de Chomsky: G3 > G2 > G1 > G0

Tipo 0: G0 Lenguaje sin restricciones.

Cualquier cosa \rightarrow Cualquier cosa.

$aBCa \rightarrow aCBa$

$aBa \rightarrow a^k a$

Regla compresora $bC ::= b$ $|izq| > |dcha|$

Estructura de frases: $AS \rightarrow SA$

No conserva el contexto ($A \rightarrow x$, $As \rightarrow xs$)

Tipo 1: Sensibles al contexto G1

Es una G0 con estructura de frases, pero sin reglas compresoras, excepto $S \rightarrow \lambda$ (Axioma)

Conserva el contexto. $aSb \rightarrow abb$

$|izq| \leq |dcha|$

Tipo 2: G2 De contexto libre.

$NT \rightarrow$ cualquier cosa. A la izquierda un solo símbolo no terminal

Si $\lambda \in L \Rightarrow S \rightarrow \lambda$, no válido si no es el axioma.

Se puede cambiar en cualquier contexto notable

Tipo 3: G3 Gramáticas regulares.

Líneales por la izquierda: $A \rightarrow aB$ $NT \rightarrow T$ ó $NT \rightarrow I$ Solo en un lado el T

Líneales por la derecha: $A \rightarrow Ba$ $NT \rightarrow NTI$ ó $NT \rightarrow TI$ Solo en el lado derecho el T

Si hay terminales en los dos lados, es G2

Gramáticas equivalentes: Si representan el mismo lenguaje. Dada una gramática lineal por la derecha cualquiera, existe otra lineal por la izquierda equivalente y viceversa.

Algoritmo: 3 pasos.

1) Construir una gramática equivalente que No sea recursiva en el axioma.

1) Identificarlos. Es decir que tras el primer paso no vuelva al axioma. $A \rightarrow aS$ Axioma inducido.

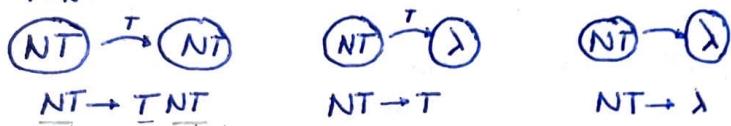
2) Añadir I. - Eliminar el axioma inducido, añadiendo un nuevo símbolo a $\Sigma_N \rightarrow I$

3) Copiar S con I. - Copiamos las reglas del axioma en I. $+ I \in \Sigma_N^*$ $I \rightarrow A/aA$

4) Eliminar axioma inducido con I. - Cambiamos en el axioma inducido, la S por I $A \rightarrow aS \Rightarrow A \rightarrow aI$
- $S \rightarrow \lambda$ No participa en este proceso.

2) Construir un grafo dirigido a partir de la gramática.

$|\Sigma_N| + 1 = N^o$ de nodos Los arcos son Terminales (Σ_T)



3) Se intercambian las etiquetas y se construye un nuevo grafo.

- Intercambiar la etiqueta S y λ $S \xrightarrow{a} \lambda \Rightarrow \lambda \xrightarrow{a} S$

- Invertir los arcos. $\lambda \xrightarrow{a} S \Rightarrow S \xleftarrow{a} \lambda$

Interpretar las reglas del grafo $S \rightarrow \lambda a; S \rightarrow a$

Árboles de derivación: A las derivaciones de las G tipo 1, 2, 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.

Es un árbol ordenado y etiquetado que se construye:

La raíz se denota por el axioma de G (S)

Una derivación directa se representa por un conjunto de ramas que salen de un nodo

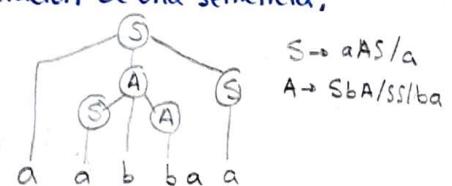
Por cada uno de los símbolos NT se dibuja una rama que sale de ese símbolo Parte 18.9 de la Producción

Nodo de partida \rightarrow padre Nodo final \rightarrow hijo Dos nodos hijos del mismo padre \rightarrow hermanos

Ascendente \rightarrow cuando es su padre Descendiente \rightarrow cuando es su hijo.

Si leemos los finales de izq. a dch da la forma sentencial, sino son todos hojas,

si son hojas todos los finales es una sentencia.



Subárbol de derivación: Dado un árbol, se llama subárbol de A al árbol:

Raíz, el propio A.

Los nodos, son los descendientes de A



Las hojas de un subárbol, leídas de izq. a dch forman una frase respecto a la raíz (NT).

Ambigüedad: Concepto relacionado con los árboles de derivación. Obtener la misma sentencia con árboles distintos.

$$\begin{array}{l} A \rightarrow AB/AA \\ B \rightarrow A \\ \hline \end{array} \Rightarrow A \rightarrow AA$$

Equivale

Niveles de ambigüedad:

Sentencia: Si puede obtenerse por medio de dos o más árboles de derivación diferentes.

Gramática: Si contiene al menos una sentencia ambigua.

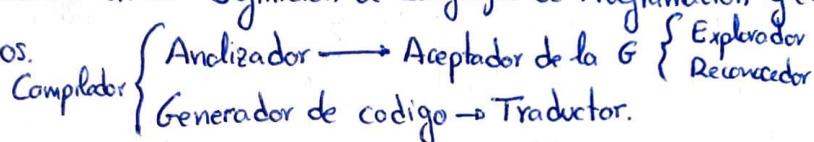
Lenguaje inherentemente ambiguo: Si todos los gramáticos que lo generan son ambiguas, no podemos encontrar una que no lo sea.

Aunque G sea ambigua el lenguaje no tiene porque serlo, si ocurre se puede encontrar una G' que no lo sea.

Es 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.

Gramáticas tipo 2: Independientes de contexto.

Son las empleadas en la Definición de Lenguajes de Programación y en la compilación de los mismos.



Se representa el lenguaje con $L(G_2)$

1) Dada una G , el lenguaje que genera, ¿es vacío o no?

Sea $G_2 : m = |\Sigma_n| = C(\Sigma_n)$, $L(G) \neq \emptyset$ si $\exists x \in L(G_2)$ 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 $\leq m = C(\Sigma_n)$ mediante:

a. todos los árboles de longitud 0.

b. A partir del conjunto de árboles de longitud n , generamos el de longitud $n+1 \leq m+1$

aplicando al conjunto de partida una producción que no haga duplicarse algún NT.

c. Se aplica b. recursivamente hasta no poder generar más árboles de longitud $\leq m$ sin repetir.

Al ser m y el n° de reglas P finito, el algoritmo termina.

Si no hay produc
Términos, finalizar
y esto es vacío

$L(G_2) = \emptyset$ si ninguno de los árboles genera una sentencia (que solo haya no terminales)

2) Si $L(G_2)$ es no vacío, comprobar si $L(G_2) = \Delta$

Construir el grafo cuyos nodos están etiquetados con los Σ_n , tal que:

- $A \xrightarrow{\alpha} B$: A con un arco hacia B.

Si no existe ciclos, $L(G_2) = \text{Finito}$.

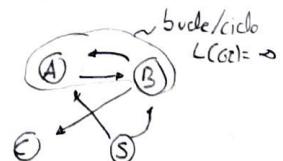
Si existen ciclos accesibles desde el axioma, $L(G_2) = \text{Infinito}$.

$$S \rightarrow aB/aA$$

$$A \rightarrow abB$$

$$B \rightarrow bC/aA$$

$$C \rightarrow c$$



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

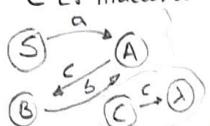
1) Limpieza de Gramáticas:

a) Reglas innecesarias: Las reglas $A \rightarrow A \in P$, hacen que sea ambigua \Rightarrow Eliminarla.

b) Símbolos inaccesibles: Sea $U \rightarrow x \in P$, donde $U \in \Sigma_N \neq S$ y no aparece en la parte derecha de ninguna otra regla de producción, U es inaccesible.

Método 1: Vector con Σ_N y Σ_T , empezar mirando S y ir marcando las que salen.

$S \xrightarrow{a} A, A \xrightarrow{b} C$
 $C \xrightarrow{c} D, B \xrightarrow{d} A$
 $\{S, A, B, C, D\}$
 C es inaccesible



Método 2: Construir el grafo y observar si no se puede llegar desde S hasta algún nodo.

c) Reglas superfluas: Son aquellas que no contribuyen a la formación de palabras $x \in \Sigma^*$.

Estas reglas contiene algún símbolo No Terminal no generativo.

Algoritmo:

- 1) Marcar los NT, si produce λ , o si contiene algún NT y alguno ya marcado, como parte derecha.
- 2) Si todos los NT están marcados \Rightarrow No existen símbolos superfluos y fin.
- 3) Si la última vez que se pasó por 1) se marcó algún NT, volver a 1)
- 4) Todo no marcado es superfluo. (Mirar el símbolo no en toda la producción)

$S \xrightarrow{a} BE, A \xrightarrow{a} CE$

$B \xrightarrow{b} Cf, C \xrightarrow{c} Cf, D \xrightarrow{d} f$

1(paso) Marcar A, D ($A \xrightarrow{a} C, D \xrightarrow{d} f$)

2(p) Marcados A, D, B ($B \xrightarrow{b} A$)

3(p) Marcados A, D, B, S ($S \xrightarrow{a} B$)

4(p) No hay más \Rightarrow Fin

$D \xrightarrow{d} f, C \xrightarrow{c} Cf$

Cro-superfluos, si no es generativo

2) Eliminación de símbolos no generativos:

Si $L(G(A)) = \emptyset \Rightarrow A$ es símbolo no generativo, y se puede eliminar, así como todas las reglas que lo contengan, obteniendo otra G2 equivalente.

Cro que son las de las reglas superfluas!

3) Eliminación de reglas no generativas: Son $A ::= \lambda$ ($A \notin \Sigma_{\text{ini}}$)

Algoritmo:

1) Eliminar de la gramática las reglas de la forma $U ::= \lambda$

2) Donde aparezca U en la parte derecha de alguna regla, $V ::= xUy$, se añade la regla $V ::= xy$ (a menos que ya exista)

3) Repetir 2) hasta que no quede ninguna a la que sustituirlo, ó $S \xrightarrow{\lambda}$

4) Eliminación de reglas de redenominación: Del tipo $A ::= B$ (donde $A \neq B$)

Algoritmo:

1) Eliminar de la gramática las reglas $U ::= V$

2) Donde aparezca la $V ::= x$, añadir $U ::= x$ (si no existe)

3) Repetir 2) hasta que no quede ninguna regla de redenominación.

$S \xrightarrow{a} PQ / aSb / P$

$P \xrightarrow{a} PQ / a$

$Q \xrightarrow{b} b / \lambda$ desaparece

\Downarrow

$S \xrightarrow{a} PQ / aSb / P$

$P \xrightarrow{a} PQ / a / a$

$Q \xrightarrow{b} b / b$

\Downarrow

$S \xrightarrow{a} PQ / aSb / aPQ / aP / a$

$P \xrightarrow{a} aPQ / aP / a$

$Q \xrightarrow{b} b / b$

Forma Normal de Chomsky (FNC)

Una gramática bien formada está en FNC si las partes derechas de todas las reglas de producción tienen al sumo dos símbolos, y cuando son dos, son NT.

$$NT \rightarrow NT \cdot NT^* \quad NT \rightarrow T \quad S \rightarrow \lambda$$

A partir de cualquier gramática de Tipo 2 se puede construir otra FNC

- 1) Bien formar la gramática.
- 2) Las que no tengan la forma $NT \rightarrow NT \cdot NT^*$ o $NT \rightarrow T$, se separa el primer elemento de la parte derecha del resto y se le asigna una letra a cada lado (si es necesario). Y desaparece lo original.

$$\begin{aligned} E_j: S \rightarrow B/a &\Rightarrow S \rightarrow BA \quad A \rightarrow a \quad S \rightarrow C/bb \Rightarrow S \rightarrow CZ \quad Z \rightarrow bb \\ S \rightarrow a/B/a/C &\Rightarrow S \rightarrow FH \quad F \rightarrow a \quad H \rightarrow B/a/C \Rightarrow H \rightarrow BI \quad I \rightarrow a/C \Rightarrow I \rightarrow AC \quad A \rightarrow a \end{aligned}$$

Forma Normal de Greibach (FNG)

En ella todas las reglas tienen la parte derecha comenzando con un terminal seguido opcionalmente de uno o varios NT.

$$NT \rightarrow T \quad NT \rightarrow T \cdot NT^*$$

- 1) Limpiar y formar bien. Eliminar la recursividad a izquierdas:

$$A \rightarrow A\alpha_1 | A\alpha_2 | B_1 | B_2 | C \Rightarrow A \rightarrow \beta_1 / \beta_2 / \beta_3 X / \beta_4 X \quad X \rightarrow x_1 / x_2 L_1 / x_3 X / x_4 L_2 X$$

$X = S^* / S \rightarrow S^* / S \rightarrow S^* / S \rightarrow S^* / S \rightarrow S^* / S$

$\text{En } X \text{ y sin } \lambda \text{ ni recursividad, tal vez } x_1, x_2, x_3, x_4$

(No se hace con λ , se pone en el principio)

Tiene que seguirstando bien formada

- 2) Aplicar el algoritmo de transformación a FNG

1. Establecer el orden de los NT, el que más convenga al 2. A, B, C

2. Clasificar en grupos las reglas:

Grupo 1) Los que ya tengan la forma. $B \rightarrow a \quad C \rightarrow aCB$

Grupo 2) Cuando la parte izquierda va antes que el NT primero de la derecha. $B \rightarrow aD$

Grupo 3) Cuando la parte derecha precede a la izquierda. $C \rightarrow BA$

3. Transformar las reglas. Grupo 3 \rightarrow Grupo 2 \rightarrow Grupo 1

Se sustituye la primera NT de la derecha (grupos 3 y 2) por todas sus reglas. Mirar si se convierten en recursivas a izquierdas.

Si convierten
se transforman

Repetir para todas las reglas del grupo 3, y posteriormente para las del grupo 2 o se han transformado en otra de Grupo 3/2

Tema 5: Lenguajes Regulares.

Sea el AF, $A = (\Sigma, Q, q_0, f, F)$, existe una G3 LD tal que $L(G3LD) = L(A)$

G3LD \rightarrow AF

Construir una gramática G3LD a partir del Automata AF:

G3LI \rightarrow G3LD
↓
AF

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

$$AF = (\Sigma, Q, q_0, f, F)$$

↓ ✓ ✗ / ↗
Se añade un simbolo nuevo F

Primero la gramática debe estar limpia y bien formada.

$$P \Rightarrow f : A + aB \rightsquigarrow f(A, a) = B \quad A \rightsquigarrow a \rightsquigarrow f(A, a) = F \quad S \rightsquigarrow \lambda \rightsquigarrow f(S, \lambda) = F$$

Cuando no hay estado destino va al final.

AF \rightarrow G3LD
↓
G3LI

Construir un Automata finito a partir de una gramática.

$$AF = (\Sigma, Q, q_0, f, F)$$

Tras pasarlo a gramática, limpiarla y bien formarla.

$$G = (\Sigma_T, \Sigma_N, S, F)$$

$$f \Rightarrow P : f(p, a) = q \rightsquigarrow p \rightarrow aq \quad f(p, a) = F \rightsquigarrow p \xrightarrow{\text{Añadir final}} a/F$$

$$f(p, \lambda) = q \rightsquigarrow p \rightarrow q \quad f(p, \lambda) = F \rightsquigarrow p \rightarrow \lambda/F$$

Expresiones Regulares: Metalenguaje para expresar el conjunto de palabras aceptadas por un AF. (Lenguajes tipo 3 o regulares)

Dadas los elementos/símbolos: $\Sigma, \emptyset, \lambda$ y las operaciones: $+$ (unión), \cdot (concatenación)

y * (cierre o clausura).

\emptyset es una ER λ es una ER $\forall a \in \Sigma$ es una ER

Si α y β son EERR $\Rightarrow \alpha \cdot \beta$ y $\alpha + \beta$ son EERR α ER $\Rightarrow \alpha^*$ es ER.

Son expresiones regulares las obtenidas al aplicar las reglas un número finito de veces.

Prioridad de las operaciones: $* > \cdot > +$

Cada ER describe o expresa un lenguaje regular.

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

$$1) (\alpha + \beta) + \gamma = \alpha + (\beta + \gamma) \quad 2) \alpha + \beta = \beta + \alpha \quad 3) \alpha \cdot (\beta + \gamma) = (\alpha \cdot \beta) + (\alpha \cdot \gamma)$$

$$4) \alpha \cdot (\beta + \gamma) = (\alpha \cdot \beta) + (\alpha \cdot \gamma) \quad 5) \alpha \cdot \lambda = \lambda \cdot \alpha = \alpha \quad 6) \alpha \cdot \emptyset = \emptyset \cdot \alpha = \emptyset \quad 7) \lambda^* = \lambda$$

$$8) \alpha \cdot \emptyset = \emptyset \cdot \alpha = \emptyset \quad 9) \alpha^* \cdot \alpha^* = \alpha^* \quad 10) \alpha \cdot \alpha^* = \alpha^* \quad 11) \lambda + \alpha \cdot \alpha^* = \alpha^*$$

$$12) (\alpha^*)^* = \alpha^* \quad 13) (\alpha^* + \beta)^* = (\alpha^* \cdot \beta^*)^* = (\alpha + \beta)^*$$

Ojo) $(ab + cb) = (a+c)b$

El & b que esta se saca.

Reglas de inferencia:

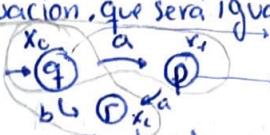
Dados tres EERR (L, A, B), sea la ecuación $\underline{L = AL + B}$, $L = AL + \emptyset$
 donde $\lambda \notin A$, entonces se verifica que $\underline{L = A^*B}$ $L = A^* \emptyset = \emptyset$

Teorema de análisis de Kleene:

Encontrar el lenguaje asociado a un determinado AF, mediante su expresión regular. $AF \rightarrow Ec. \text{ característica} \rightarrow ER$

1) Escribir la ecuación característica:

Los nodos etiquetados con x_i

Cada nodo da lugar a una ecuación, que será igual al nodo destino (x_i) por el símbolo para llegar (a_{ij}).  $x_0 = ax_1 + bx_f$
 $x_1 = x_2a$
 $x_2 = \lambda$

Si va hacia un nodo final, se añade también el símbolo de transición solo  $x_0 = bx_1 + b$

2) Resolver las ecuaciones, de más alejada a más cercana al inicio.

Utilizando las reglas, sobre todo las de inferencia. $L = AL + B$ $L = A^*B$

3) El resultado será la x_0 que identifica a q_0

17/11/2019

Teorema de síntesis de Kleene:

Encontrar un reconocedor (AF) para un lenguaje regular dado.

- Derivar la ER y obtener una G3LD y de ella un AF.

Derivada de una ER $\Rightarrow D_a(ER)$ = Quitar de la cabeza de la ER una ' a '

$$D_a(\emptyset) = \emptyset$$

$$D_a(R+S) = D_a(R) + D_a(S)$$

$$D_a(\lambda) = \emptyset$$

$$D_a(R \cdot S) = D_a(R)S + \delta(R)D_a(S) \quad \forall R$$

$$D_a(a) = \lambda \quad a \in \Sigma$$

$$D_a(R^*) = D_a(R) \cdot R^*$$

$$D_a(b) = \emptyset \quad b \neq a, b \in \Sigma$$

$$\delta(R) = \lambda \text{ o } \emptyset \quad \text{Si } R \text{ puede ser } \lambda, \text{ es lambda sino vacio.}$$

$$D_a b(R) = D_b(D_a(R))$$

Hallar todas las derivadas con todos los símbolos de la expresión y sus deltas. (Finitas)

Transformar derivadas en reglas de producción.

$$D_a(R) = S \Rightarrow R \xrightarrow{\text{cig.}} aS/a \quad \text{Mirar delta para ver si es } \emptyset \text{ o } \lambda$$

$$D_a(R) = S \Rightarrow R \xrightarrow{\text{cig.}} aS/a \quad \text{Si } S \neq \emptyset \text{ y } \delta(S) = \lambda / D_a(R) = a \quad \text{Si } S = \emptyset \text{ y } \delta(S) = \lambda$$

Si dan vacío no se usan en ninguna regla en P.

El axioma es lo original R_0 :

Σ_T = Símbolos que forman R_0

Σ_{NT} = Las letras de las distintas derivadas.

Dada una ER que representa a un lenguaje regular, construir un AF que acepte ese lenguaje regular. Sea α una Expresión Regular:

$$\alpha = \emptyset$$



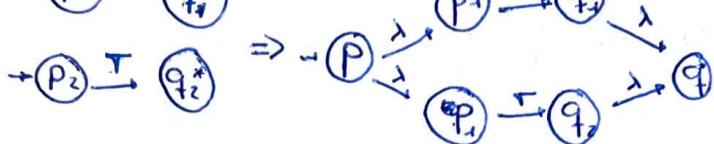
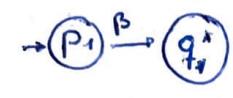
$$\alpha = a, a \in \Sigma$$



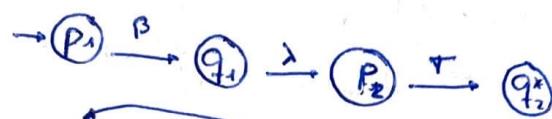
$$\alpha = \lambda$$



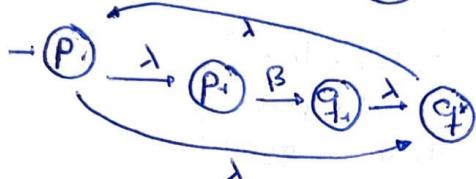
$$\alpha = B + T$$



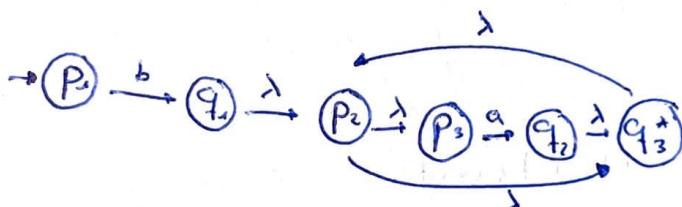
$$\alpha = B \cdot T$$



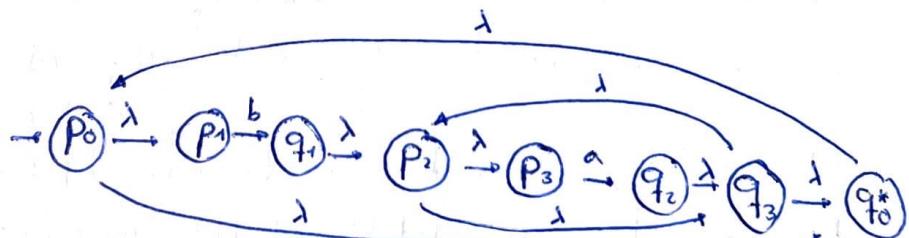
$$\alpha = B^*$$



$$\alpha = b \cdot a^*$$



$$\alpha = (b \cdot a^*)^*$$



Tema 6: Autómatas a Pila.

24/11/2019

AF + Pila $\xrightarrow{AP_V}$ Acepta si la pila está vacía

$\xrightarrow{AP_F}$ Acepta si llega a un estado final, independientemente de la pila.

Para cada gramática G independiente del contexto, existe un automata de pila

M tal que: $L(G) = L(M)$ y también al revés.

Existe un lenguaje independiente del contexto que no es el lenguaje aceptado por ningún autómata de pila determinista.

Se puede:

No se puede pasar un APND a APD

G2 FNG \rightarrow APV D

APV \rightarrow G2

Definición:

Hay una cinta/palabra que va pasando y un control de estado que los va leyendo de pila y según el símbolo de la palabra apila o desapila.
El símbolo leído desaparece de pila al ser un lenguaje de pila.

Definición formal:

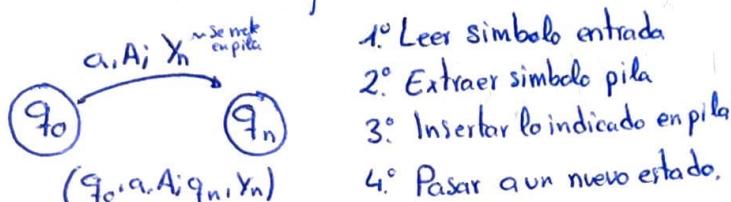
$$\begin{array}{ll} \text{AF: } \Sigma, Q, q_0, f, F & M: \text{Alfabeto de pila} \\ \left. \begin{array}{c} \Sigma \\ Q \end{array} \right\}, \left. \begin{array}{c} q_0 \\ f, F \end{array} \right\} & A_0: \text{Símbolo inicial de la pila} \\ \text{AP: } \Sigma, \Gamma, Q, A_0, q_0, f, F & \end{array}$$

Transición:

$$f(q_0, a, A) = \{(q_1, z_1), (q_2, z_2), \dots\}$$

Estado $\xrightarrow{\text{Cima pila}} \downarrow$ Cima pila
 $\xrightarrow{\text{Símbolo leído}} \downarrow$ Símbolo leído

$\xrightarrow{\text{Se ha quitado } A \text{ y se apila } z_i}$
 Estado al que salta.



Función de transición:

$$f: Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow P(Q \times \Gamma^*) \quad Q \times \Sigma \times \Gamma$$

Transiciones dependientes de la entrada $\rightarrow f(q, a, A) = (q, AA)$

Transiciones independientes de la entrada $\rightarrow f(q, \lambda, A) = (q, AA)$

$$Q \times \lambda \times \Gamma$$

Determinista \rightarrow Solo da lugar a una transición (q, z_i) y no puede haber para el mismo símbolo de entrada dependientes e independientes.

No determinista \rightarrow Posibilidades múltiples $\{(q_1, z_1), (q_2, z_2), \dots\}$

\hookrightarrow Si para el mismo símbolo hay transiciones dependientes e independientes.

Descripción instantánea.

Permite describir sencillamente la configuración del AP en cada momento.

Terna (q, x, z) $q \in Q, x \in \Sigma^*, z \in \Gamma^*$

q : Estado actual
 x : Palabra leída
 z : Lo que hay en pila

Movimiento: $(q, ay, Ax) \xrightarrow{} (p, y, zx)$ 1 solo paso

Sucesión de movimientos: $(q, ay, Ax) \xrightarrow{*} (p, y, zx)$ Más de un paso.

Lenguaje aceptado por un AP:

AP_V: Por vaciado de pila = $\{x / (q_0, x, A_0) \xrightarrow{*} (p, \lambda, \lambda), p \in Q, x \in \Sigma^*\}$

AP_F: Por estadio final = $\{x / (q_0, x, A_0) \xrightarrow{*} (p, \lambda, X), p \in F, x \in \Sigma^*, X \in \Gamma^*\}$

\hookrightarrow Es final.
 \hookrightarrow En vacío.

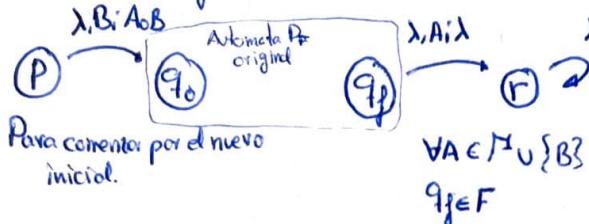
Equivalecias:

Para cada autómata de pila que acepte cadenas sin vaciar su pila, existe un automata equivalente pero que vacia su pila antes de llegar a un estado de aceptación.

$$AP_F \rightarrow AP_V \quad (\Sigma, M, Q, A_0, q_0, f, F) \rightarrow (\Sigma, M \cup \{\beta\}, Q \cup \{p, r\}, p, f', \emptyset)$$

Nuevo simbolo en pila y dos estados.

Sin estado final. \hookrightarrow Nuevo inicial \hookrightarrow es el nuevo inicial.



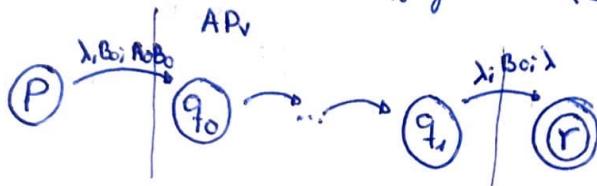
Para vaciar la pila cuando el otro terminaba.

El resto de transiciones son las de f.

solo que uniendo la inicial y los finales.

1/12/19

$$AP_V \rightarrow AP_F \quad (\Sigma, M, Q, A_0, q_0, f, \emptyset) \rightarrow (\Sigma, M \cup \{\beta\}, Q \cup \{p, r\}, p, f', \emptyset)$$



Introduce el nuevo simbolo de pila en la base y se saca al final para llevar a un estado final.

$$G2FNG \rightarrow AP_V \quad (\Sigma_T, \Sigma_N, S, P) \rightarrow (\Sigma = \Sigma_T, M = \Sigma_N, Q = q, A_0 = S, p_0 = q, f = P, \emptyset)$$

Las funciones de transición:

Solo hay un estado.

$$A \xrightarrow{\alpha} \overline{BCD} \Rightarrow f(q, \overline{\alpha}, A) \rightarrow (q, \overline{BCD})$$

$$A \xrightarrow{\beta} \overline{b} \Rightarrow f(q, b, A) \rightarrow (q, \lambda)$$

Cima Simbolo entrada

A lo que da lugar.

$$AP_V \rightarrow G2 \quad (\Sigma, M, Q, A_0, q_0, f, F) \rightarrow (\Sigma_T = \Sigma, \Sigma_N = \{S, \text{Estado pila}\}, S, P)$$

Ejem:

$$\Sigma = \{S, pAp, pAq, qAp\}$$

Todas las posibilidades.

$S \rightarrow q_0$ A0 Estado. Todas las posibles.

Molde de las funciones de transición: Tras hacer las transiciones cambiar la notación

$$f(q, \alpha, A) = (p, \overline{BCD}) \quad \text{Casos 1}$$

de los NoTerminales $qAp = A$
Sustituir, limpiar y bien formar.

$$qAp \rightarrow \overline{\alpha} \overline{BCD} \quad \text{Casos 2}$$

Rellenar los huecos con las combinaciones posibles. $qAp \rightarrow \overline{\alpha} \overline{BCD} \quad qAp \rightarrow \overline{\alpha} \overline{CD}$

$$f(q, \alpha, A) = (q, \lambda) \quad \text{Casos 2}$$

Tema 7: Maquinas de Turing.

Es una abstracción.

Dispositivo hipotético capaz de manipular símbolos en una tira de cinta considerando ciertas reglas. Puede simular la lógica de cualquier algoritmo de un computador.

Formada por:

Cinta infinita dividida en celdas.

Cabecilla de lectura/escritura capaz de moverse por la cinta de izq. a dch.

Operaciones:

1. Pasa a un nuevo estado.
2. Escribe un nuevo símbolo en la cinta. (reemplaza)
3. Mueve el cabecilla de L/E hacia dch., izq o se para.

Definición formal:

$$MT: (\Sigma, \Gamma, b, Q, q_0, f, F)$$

$\Sigma \subseteq \Gamma$ Alfabeto de entrada.

Γ Alfabeto de símbolos de la cinta. Mas amplio incluye λ

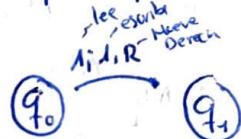
$b \in \Gamma$ Símbolo especial - espacio en blanco ($b \notin \Sigma$)

Q Conjunto estados.

$q_0 \in Q$ Estado inicial.

f función $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L(-), R(+), S(=)\}$ $f(q, a) \rightarrow (p, b, L)$

F Conjunto estados finales.



Cinta infinita por ambos lados.

Nº finitos de símbolos entrada.

La cabeza se inicia a la derecha del primer blanco. $b \neq \lambda$

$$f(q, a, A) = (q, \lambda) \quad f(q, \lambda, S) = (q, \lambda) \quad f(p, a, A) = (q, b)$$

(q A q → a) (q S q → λ)

↓ ↓

ojo []

$$\begin{aligned} p A r &\rightarrow a B r \\ p A p &\rightarrow a q B p \\ p A q &\rightarrow a q B q \end{aligned}$$

Hay que probar todos los posibles →

Ejemp:

$$APr = \{ \{a, b\}, \{A, B\}, \{p, q\}, A, p, f, \varnothing \}$$

$$G_1 (\Sigma_r = \{a, b\}, \Sigma_N = \{S, \overset{A}{p}, \overset{B}{p}, \overset{C}{q}, \overset{D}{q}, \overset{E}{p}, \overset{F}{p}, \overset{G}{q}, \overset{H}{q}, pAp, pAq, qAq, qAp, pBp, pBq, qBq, qBap, S, P\})$$

$$1) S \rightarrow \overset{A}{p} \overset{B}{p} / \overset{B}{p} \overset{A}{q} \quad (\text{redenominación de } q_0 \text{ a } q_i)$$

$$2) f(p, a, A) = (p, BA)$$

"más" $(p A) = a (p B) (A)$

$$\begin{aligned} pAp &\rightarrow a(pBp)(pAp) / a(pBq)(qAp) \\ pAq &\rightarrow a(pBp)(pAq) / a(pBq)(qAq) \end{aligned}$$

$$A \rightarrow aEA / aFD$$

$$B \rightarrow aEB / aFC$$

Tema 7: Maquinas de Turing.

Es una abstracción.

Una cinta infinita y la cabecera de lectura se puede mover de derecha a izquierda.

También es destrutivo, lo que lee se borra, pero se puede escribir de nuevo.

Escribe y lee en cinta.

Definición formal

$$MT = (\Sigma, M, b, Q, q_0, f, F)$$

$\Sigma \times M$ inducidos $\Sigma \times \Sigma$

b simbolos especiales

$$Q \text{ estados} \quad Q \times T \rightarrow Q \times M \times \{L, R, \text{left}, \text{right}, \text{stop}, \text{eraser}, \text{blank}, \text{erase}\} ; f(q_0, O) = (q_1, 1, D)$$

Caja copia
q₀ estado inicial
F conjunto finito

Cinta infinita por ambos lados.

Inicia lueg^o con un n^o finito de simbolos de entrada, seguidos de b

La cabecera se inicia a la derecha del primer blanco, más a la izq de la palabra.

$$\begin{array}{c} 101010 \\ | \\ A \\ q_0 \end{array}$$

Diagrama de estados:

Nodos → Estados



El q_{Final} no suele tener transiciones.

48/12/19

Tipos:

Transductor: Modifica el contenido de la cinta realizando cierta función.

Reconocedor: Capaz de reconocer las palabras de un lenguaje. Si pertenece o no al lenguaje.

Transductor: Si la entrada está bien formada terminará en un estado final.

Y si no bien formada no acaba en uno final.

No para sobre un blanco, sino que vuelve al comienzo de la cadena.

* Siempre tiene que acabar, sea correcta o no.

Reconocedor: Si reconoce la palabra ~~que~~ acaba en un ^{Est.} final.

Suele poner blancos en la cinta para reconocer.

Las máquinas de Turing son equivalentes si: Ambas realizan la misma acción sobre todas sus entradas. La salida debe ser la misma. Reconocen el mismo lenguaje o llegue al final para la misma función.

Variantes:

MT con alfabeto binario: $M = \{0, 1, B\}$

MT limitada por un extremo (linealmente acotado)

MT con restricciones en el movimiento de L/E

MT Universal.

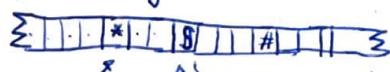
Capaz de simular el comportamiento de cualquier MT, MT "programable".
Tiene dos partes en su cinta:

1º Con la entrada de la cinta.

2º El programa, la codificación para su funcionamiento.

CONTENIDO DE LA CINTA
(Entrada) (1s y 0s)
PROGRAMA

Está codificado en binario.



La posición en la cinta.
Donde está la cabecera.

Cinta MTB

MT y Computabilidad.

La MT U ~~puede resaltar~~ cualquier problema computable.

Complejidad computacional:

Recursos: Tiempo y ...

Complejidad de un problema decidable.

El problema de la parada:

No existe una MT que sepa cuando se vaya parar otra MT.

1. RECURSOS

Limpiar y bien formar siempre

1.1. Tema 3

1.1.1. AFD → AFD mínimo

1. Buscar Q/E_0 , que divide los estados en finales y no finales.
2. Hacer $Q/E_1, Q/E_2\dots$ 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 → AFD

1. Calcular T^* .
2. Quitar la columna λ y añadir la columna λ^* .
3. Sustituir la columna a, b\dots por $\lambda^*a\lambda^*, \lambda^*b\lambda^*\dots$
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 → 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 ($A \rightarrow A$)
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 $\Sigma_{NT} \rightarrow \Sigma_T$ y los $\Sigma_{NT} \rightarrow \lambda$
 - 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 $A \rightarrow \lambda$. Cada vez que aparezca A en la λ . Se admite parte derecha, se añade la posibilidad de que sea $Axioma \rightarrow \lambda$, OJO si cuando eliminamos solo quedaba un símbolo ($C \rightarrow M$ y eliminamos M), ponemos lambda ($C \rightarrow \lambda$) y repetimos el proceso de eliminación (para C).
6. Eliminar las reglas de redenominación, las de tipo $A \rightarrow B$. Por cada regla de la forma $B \rightarrow x$, se añade $A \rightarrow x$

1.2.5. G2 → FNC

Se separa el primer símbolo de la derecha del resto, por ejemplo, de $A \rightarrow aBb$ sacamos $A \rightarrow DE$ y $D \rightarrow a$, $E \rightarrow Bb \rightarrow BC$, $C \rightarrow b$ (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 $A \rightarrow A\alpha | \beta$ (donde α y β son cualquier cosa...)

Se transforma en:

- $A \rightarrow \beta | \beta X$
- $X \rightarrow \alpha X | \alpha$

Si tuviéramos varias ($A \rightarrow A\alpha | \beta_1 | \beta_2$), entonces se transforma en:

- $A \rightarrow \beta | \beta X$
- $X \rightarrow \alpha X | \alpha$

1.2.8. Paso de G3LD (FNG) \rightarrow 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. Teoría de síntesis

Nota: $D_{ab}(\alpha) = D_b(D_a(\alpha))$

- 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

- Si R_x puede ser λ , se añade una regla, $R_x \rightarrow \lambda$
- Si $D_y(R_{x_1}) = R_{x_2}$, se añade una regla $R_{x_1} \rightarrow yR_{x_2}$
- Luego se aplica el paso de G3LD (FNG) a AF para obtener el AF correspondiente

$$D_a(a) = \lambda$$

$$D_a(b) = \emptyset$$

$$D_a(RS) = D_a(R)S + d(R)D_a(S)$$

$$D_a(R + S) = D_a(R) + D_a(S)$$

$$D_a(R^*) = D_a(R)R^*$$

$$d(a) = \emptyset$$

$$d(a^*) = \lambda$$

Si puede ser λ , es λ , si no \emptyset

$$d(a^* + a) = \lambda$$

1.4. Tema 6

1.4.1. APF → APV

- Añadir un estado inicial nuevo con una transición y un nuevo «chivato» que nos indique cuando se vacía la pila
- Añadir un estado «final» que desapile todo lo que pudiera quedar y el chivato.

1.4.2. APV → APF

- Añadir un estado inicial nuevo con su chivato
- Añadir un estado final al que se transita desapilando el chivato

1.4.3. $G2(FNG) \rightarrow APV$

1. Tres tipos de reglas:

- a) $A \rightarrow aBCD : f(q, a, A) = (q, BCD)$
- b) $A \rightarrow a : f(q, a, A) = (q, \lambda)$
- c) $S \rightarrow \lambda : f(q, \lambda, S) = (q, \lambda)$

2. El automata tendrá un único estado, q.

1.4.4. $APV \rightarrow G2$

1. Se pone una regla del tipo $S \rightarrow (q_0, A_0, pqr\dots)$

2. Se ponen reglas de tipo:

- a) Tipo 1A : $f(q, a, B) = (p, DEF)$
El molde será: $(qB_ \rightarrow a(pD_) (_E_) (_F_))$
- b) Tipo 1B : $f(p, \lambda, B) = (q, A)$
El molde será: $(pB_ \rightarrow X(qA_))$
- c) Tipo 2A : $f(p, a, B) = (q, \lambda)$
El molde será: $(pBq) \rightarrow a$
- d) Tipo 2B : $f(p, \lambda, B) = (q, \lambda)$
El molde será: $(pBq) \rightarrow \lambda$

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. $\alpha^* = \lambda + \alpha + \alpha^2 + \dots + \alpha^n + \alpha^{n+1} \cdot \alpha^*$
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:

$$f(\alpha, \beta, \dots, \sigma) + (\alpha + \beta + \dots + \sigma)^* = (\alpha + \beta + \dots + \sigma)^*$$
17. Sea f una función, $f : E_{\Sigma^n} \rightarrow E_\Sigma$ se verifica:

$$(f(\alpha^*, \beta^*, \dots, \sigma^*))^* = (\alpha + \beta + \dots + \sigma)^*$$
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. Análisis

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 $NT \rightarrow T\ NT$