

# INF155: Informática Teórica

Aldo Berrios Valenzuela

19 de julio de 2016



# Índice general

<b>I Teoría de Autómatas y Lenguajes Formales</b>	<b>7</b>
<b>1. Preliminares</b>	<b>9</b>
1.1. Preámbulo	9
1.1.1. Alfabetos, palabras, lenguajes	9
1.1.2. Operaciones entre palabras	9
1.1.3. Operaciones entre lenguajes	10
1.2. Expresiones Regulares	11
1.3. Ejercicios	12
<b>2. Autómatas Finitos</b>	<b>15</b>
2.1. Autómatas Finitos Deterministas (DFA)	15
2.2. Autómatas Finitos No Deterministas (NFA)	17
2.3. Relaciones entre DFA, NFA y Expresiones Regulares	18
2.3.1. Obtención de un NFA a partir de una expresión regular	18
2.3.2. Obtención de un DFA a partir de un NFA	20
2.3.3. Obtención de una Expresión Regular a partir de un DFA	26
2.4. DFA Mínimo	28
<b>3. Propiedades de los Lenguajes Regulares</b>	<b>35</b>
3.1. Lema de Bombeo	35
3.1.1. Estrategia de demostración usando el Lema de Bombeo	37
3.2. Propiedades de Clausura	37
3.3. Un lenguaje que cumple el Lema de Bombeo sin ser regular	40
3.4. Ejercicios	40
<b>4. Pushdown Automata</b>	<b>41</b>
4.1. Definición de un Autómata de Stack	41
4.1.1. Notación Gráfica de un Autómata de Stack	41
4.1.2. Descripción instantánea y lenguajes de un PDA	42
<b>5. Gramáticas y Lenguajes de Contexto Libre</b>	<b>45</b>

5.1. Gramáticas . . . . .	45
5.1.1. Jerarquía de Chomsky . . . . .	45
5.2. Gramáticas de Contexto Libre . . . . .	47
5.2.1. Ejemplo informal de una Gramática de Contexto Libre . . . . .	47
5.3. Árboles de Derivación . . . . .	48
5.4. Ambigüedad en Gramáticas y Lenguajes . . . . .	50
5.5. Equivalencia entre PDA y CFG . . . . .	52
5.5.1. Transformar una CFG a un PDA . . . . .	52
5.5.2. Transformar un PDA a una CFG . . . . .	53
<b>6. Propiedades de los Lenguajes de Contexto Libre</b>	<b>55</b>
6.1. Forma normal para Gramáticas de Contexto Libre . . . . .	55
6.1.1. Eliminando símbolos inútiles . . . . .	55
6.1.2. Eliminando producciones $\epsilon$ . . . . .	57
6.1.3. Forma Normal de Chomsky (CNF) . . . . .	58
6.2. Lema de Bombeo para Gramáticas de Contexto Libre . . . . .	61
6.3. Propiedades de Clausura para CFL . . . . .	62
6.4. Ejercicios . . . . .	64
<b>II Computabilidad y Problemas Intratables</b>	<b>65</b>
<b>7. Computabilidad</b>	<b>67</b>
7.1. Problemas que los computadores no pueden resolver . . . . .	67
7.1.1. El hipotético detector de “Hola Mundo” . . . . .	67
7.1.2. Reducir un problema a otro . . . . .	69
7.2. Máquinas de Turing . . . . .	71
7.2.1. Técnicas de Programación para Máquinas de Turing . . . . .	71
7.2.2. Simular un Procesador . . . . .	72
<b>8. Indecibilidad</b>	<b>75</b>
8.1. Máquina de Turing Universal . . . . .	75
8.1.1. El lenguaje diagonal . . . . .	76
8.2. Demostración de lenguajes recursivos y R.E. usando cajitas . . . . .	79
8.3. Reducción de Problemas . . . . .	81
8.4. Teorema de Rice . . . . .	82
8.5. Problema de Correspondencia de Post (PCP) . . . . .	83
<b>9. Problemas Intratables</b>	<b>85</b>
9.1. P vs NP . . . . .	85
9.1.1. Máquinas de Turing Deterministas vs otros modelos . . . . .	85

9.1.2. Reducción de problemas en tiempo polinomial . . . . .	86
9.1.3. Definición de un problema NP-completo . . . . .	86
9.2. Algunos problemas NP-completos . . . . .	87
9.2.1. Satisfiability problem (SAT) . . . . .	88
9.2.2. Independent Set (IS) . . . . .	89
9.2.3. Node Cover . . . . .	90
9.2.4. ¿Por qué hacemos estas reducciones? . . . . .	92



## **Parte I**

# **Teoría de Autómatas y Lenguajes Formales**





# Capítulo 1

## Preliminares

### 1.1. Preámbulo

---

**Idea** Teoría de lenguajes autómatas, computabilidad, complejidad.

#### 1.1.1. Alfabetos, palabras, lenguajes

**Definición 1.1.** Un alfabeto es un **conjunto finito** de símbolos atómicos. Suelen nombrarse con símbolos griegos mayúsculas:  $\Sigma, \Gamma, \Delta, \dots$

**Definición 1.2.** Una palabra (string) sobre el alfabeto  $\Sigma$  es una secuencia finita de símbolos de  $\Sigma$ . Suelen anotarse  $\alpha, \beta, \dots$ . La secuencia vacía (largo 0) se llama  $\epsilon$ . El largo de la palabra  $\alpha$  se anota  $|\alpha|$  (a veces  $\lg \alpha$ )

**Definición 1.3.** Un lenguaje sobre el alfabeto  $\Sigma$  es un conjunto de palabras sobre  $\Sigma$ .

#### 1.1.2. Operaciones entre palabras

**Definición 1.4.** Sea  $\Sigma$  un alfabeto,  $\alpha = a_1 a_2 \dots a_m, \beta = b_1 b_2 \dots b_n$  palabras sobre  $\Sigma$ . La concatenación entre  $\alpha$  y  $\beta$  es:

$$\alpha \cdot \beta = a_1 a_2 \dots a_m b_1 b_2 \dots b_n$$

Comúnmente se anota  $\alpha\beta$ .

Nótese que la concatenación no es conmutativa, en general  $\alpha \cdot \beta \neq \beta \cdot \alpha$ .

Es asociativa:

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

Vemos que:

$$\epsilon \cdot \alpha = \alpha \cdot \epsilon = \alpha$$

Además:

$$|\alpha \cdot \beta| = |\alpha| + |\beta|$$

(Notar que  $\lg \alpha \cdot \beta = \lg \alpha + \lg \beta$  para aficionados a los logaritmos... se cumple la misma propiedad)

Definimos, para  $n \in \mathbb{N}_0$ :

$$\begin{aligned}\alpha^0 &= \epsilon \\ \alpha^{n+1} &= \alpha^n \cdot \alpha\end{aligned}$$

Ojo: abuso de notación: Sea  $\Sigma$  un alfabeto,  $a$  un símbolo en  $\Sigma$ . [Hay que tener cuidado con la ambigüedad:  \$a\$  puede ser un símbolo o una palabra. Es importante identificar el contexto.](#)

Usaremos simplemente  $a$  para anotar el símbolo y también para la palabra de un símbolo (palabra de largo 1).

De la misma forma, resulta cómodo anotar  $\Sigma$  para el lenguaje de todas las palabras de largo 1 sobre  $\Sigma$ .

### 1.1.3. Operaciones entre lenguajes

Los lenguajes son conjuntos, en consecuencia, las operaciones entre conjuntos también se aplican a los lenguajes.

Los lenguajes son conjuntos de palabras. Sean  $L_1, L_2$  lenguajes sobre  $\Sigma$ :

**Definición 1.5.**  $L_1 \cdot L_2 = \{\alpha \cdot \beta : \alpha \in L_1 \wedge \beta \in L_2\}$

Notar que  $L_1 \cdot L_2 \neq L_2 \cdot L_1$ , en general.

Vemos que:  $(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3) \rightarrow$  por la asociatividad de concatenación entre palabras.

Tenemos, por propiedades de unión de conjuntos, si  $L_1, L_2, L_3$  son lenguajes sobre  $\Sigma$ :

- $L_1 \cup L_2 = L_2 \cup L_1$
- $(L_1 \cup L_2) \cup L_3 = L_1 \cup (L_2 \cup L_3)$ .
- $\emptyset \cup L_1 = L_1 \cup \emptyset = L_1$
- $(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3)$
- $\{\epsilon\} \cdot L_1 = L_1 \cdot \{\epsilon\} = L_1$
- $L_1 \cdot (L_2 \cup L_3) = (L_1 \cdot L_2) \cup (L_1 \cdot L_3)$
- $(L_1 \cup L_2) \cdot L_3 = (L_1 \cdot L_3) \cup (L_2 \cdot L_3)$

Resulta también:

$$\emptyset \cdot L_1 = L_1 \cdot \emptyset = \emptyset$$

Cuidado: no confunda  $\emptyset$ ,  $\{\epsilon\}$ ,  $\epsilon$ .

**Definición 1.6.** Sea  $L$  un lenguaje sobre  $\Sigma$ . Definimos:

- $L^0 = \{\epsilon\}$
- $L^{n+1} = L^n \cdot L \quad \text{si } n \geq 0$
- $L^* = \bigcup_{n \geq 0} L^n$
- $L^+ = \bigcup_{n \geq 1} L^n$

En particular abuso:  $\Sigma^*$  son todas las palabras sobre  $\Sigma$ .

**Observación** Aplicando las definiciones anteriores, se tiene que:

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

## 1.2. Expresiones Regulares

**Idea:** Aprovecho de las operaciones entre lenguajes...

**Definición 1.7.** Sea  $\Sigma$  un alfabeto. Definimos expresiones regulares (R.E.) y los lenguajes que denotan mediante los siguientes:

- i.  $\emptyset$  denota  $\emptyset$
- ii.  $\epsilon$  denota  $\{\epsilon\}$
- iii. Para  $a \in \Sigma$ ,  $a$  denota  $\{a\}$ .

Sean  $R, S$  expresiones regulares que denotan  $L_R$  y  $L_S$ .

- iv.  $(R) \cdot (S)$  denota  $L_R \cdot L_S$ .
- v.  $(R) \mid (S)$  denota  $L_R \cup L_S$ .
- vi.  $(R)^*$  denota  $L_R^*$  (son cero mas cosas que se pueden obtener de  $R$ ).

Extraoficialmente:

- $(R)^n$  con  $n \in \mathbb{N}_0$  denota  $L_R^n$ .
- $(R)^+$  denota  $L_R^+ = L_R \cdot L_R^*$ .

**Nota:** En el fondo las R.E. son reglas de construcción de lenguajes

**Notación:** Evitar  $()$ , dejarlos sólo “cuando necesario”.

- Consideramos  $()^n, ()^*, ()^+$  como “potencias”, es lo que primero se hace.
- Consideramos  $\cdot$  como “producto”, segunda precedencia. Comúnmente se omite.
- Consideramos  $\mid$  como “suma”, precedencia más baja.
- Sabemos que  $\cdot$  y  $\mid$  son **asociativos**, no importa el orden.

Ejemplos:

- $a^*b^*$ : son todas las palabras que comienzan con 0, 1 o muchas  $a$ 's, y enseguida continúa con 0, 1 o muchas  $b$ 's.
- $(a \mid b)^*$ : son todas las combinaciones que podemos formar con  $a$ 's y con  $b$ 's.
- $a \mid b^*$ : palabras que pueden ser 1  $a$ , muchas  $b$ 's o una palabra vacía ( $\epsilon$ ).
- $a(a \mid b)^*b$ : todas las palabras sobre  $\{a, b\}$  que comiencen con  $a$  y terminen con  $b$ .

Algunas propiedades:

- $(R^*)^* = R^*$
- $R^* = R^+ \mid \epsilon$
- $R^*R^* = R^*$
- $R^+R^* = R^+$
- $R^*R^+ = R^+$

**Teorema 1.1.** Sea la ecuación entre lenguajes sobre  $\Sigma$ :

$$X = A \cdot X \mid B \tag{1.2.1}$$

dados  $A, B \subseteq \Sigma^*$ . La solución a la ecuación de lenguajes (1.2.1) viene dada por:

$$X = A^*B \tag{1.2.2}$$

**Idea de Demostración:** Hallar una aproximación...

Sabemos que  $B \subseteq X$ . Si tomamos  $B$  y substituimos al lado derecho:  $X^{(1)} = AB \cup B$ . Nuevamente:  $X^{(2)} = A^2B \cup AB \cup B$ . Al final nos queda:

$$\rightarrow X = A^*B \quad (1.2.3)$$

Veamos:

$$\begin{aligned} A(A^*B) \mid B &= AA^*B \mid B \\ &= A^+B \mid B \\ &= (A^+ \mid \epsilon)B \\ &= A^*B = X \end{aligned}$$

O sea,  $X = A^*B$  es una solución. ¿Hay más?

Si  $\epsilon \in A$ , cualquier conjunto que incluye  $A^*B$  **es solución** (elija  $\epsilon \cdot X = X$  como primer término).

Si  $\epsilon \notin A$ : sospechamos solución única...

*Demostración.* Por contradicción. Supongamos que  $\alpha \in X$ ,  $\alpha \notin A^*B$ . Entonces:

$$\alpha \in A \cdot X \cup B$$

Sabemos  $\alpha \notin B \subseteq A^*B$ . Por lo tanto,  $\alpha \in A \cdot X$ . O sea,  $\alpha = uv$ , con  $u \in A$  y  $v \in X$ . Como  $\epsilon \notin A$ ,  $|u| > 0$ ,  $|v| < |\alpha|$ , y podemos repetir el mismo razonamiento con  $v \notin A^*B$ . Esto es absurdo, no podemos acortar indefinidamente.  $\square$

### 1.3. Ejercicios

**Ejercicio 1.1.** Escriba la expresión regular que acepta cadenas del alfabeto  $\{0, 1\}$  que terminan en 1 y los 0's siempre están seguidos de al menos dos 1's.

*Solución.* La respuesta es:

$$(011 \mid 1)^* 1$$

$\square$

**Ejercicio 1.2.** Escriba la expresión regular de los strings que comienzan con *aba* y terminan con *baca*.  $\Sigma = \{a, b, c\}$ .

*Solución.* Básicamente, la expresión regular que buscamos es:

$$abaXbaca \quad (1.3.1)$$

donde  $X$  corresponde a la expresión regular que representa un "cualquier cosa". Para el alfabeto  $\Sigma$  que tenemos, se tiene que

$$X = (a \mid b \mid c)^*$$

Al reemplazar  $X$  sobre (1.3.1) se obtiene:

$$aba(a \mid b \mid c)^* baca \quad (1.3.2)$$

A simple vista podríamos decir que estamos listos. Sin embargo, (1.3.2) rechaza la palabra *abaca*, string que comienza con *aba* y termina con *baca*. Para solucionar este problema simplemente unimos *abaca* a la expresión regular (1.3.2). Entonces, el resultado final es:

$$aba(a \mid b \mid c)^* baca \mid abaca$$

$\square$

**Ejercicio 1.3.** Dado al alfabeto  $\Sigma = \{a, b, c\}$ , construya una expresión regular que acepte todos los strings que contienen al menos una  $a$  y una  $b$ .

*Respuesta.*

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

□

**Ejercicio 1.4.** Construya una expresión regular de todas las palabras en las que aparecen las  $a$ 's antes de las  $c$ 's con  $\Sigma = \{a, b, c\}$ .

*Respuesta.*

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

□

**Ejercicio 1.5.** Construya una expresión regular de todas las palabras que comiencen en  $ab$  y terminen en  $bc$ .

*Respuesta.*

$$ab(a|b|c)^* bc|abc$$

□

**Ejercicio 1.6.** Construya una expresión regular para todas las palabras que comiencen en  $a$ , contengan  $b$  exactamente 2 veces y terminan en  $c$ .

*Respuesta.*

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

□

**Ejercicio 1.7.** Construya la expresión regular para el lenguaje de cadenas de largo impar con  $\Sigma = \{0, 1\}$ .

*Respuesta.* Buscamos todas las combinaciones posibles:

- 000
- 001
- 010
- 100

Por lo tanto, la respuesta es:

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

□

**Ejercicio 1.8.** ¿Las palabras del lenguaje español generan un lenguaje finito?

*Respuesta.* La RAE tiene todas las palabras que forman el lenguaje español. Dicho diccionario está compuesto por un conjunto finito de palabras, en consecuencia, el lenguaje español es finito. □

**Ejercicio 1.9.** ¿“El cantar del Mio Cid” es un lenguaje regular?

*Respuesta.* “El cantar del Mio Cid” es un conjunto finito de palabras, por lo tanto, es un lenguaje regular (sólo basta con crear una expresión regular que una todas las palabras del cantar). □



## Capítulo 2

# Autómatas Finitos

**Idea:** Un computador muy rudimentario. Está en un estado, y va a otro conforme lee símbolos. Parte en un estado inicial, si luego la palabra está en un estado final acepta.

**Ejemplo 2.1.** Consideremos los comentarios en C, los cuales siempre comienzan con “/\*” y terminan con “\*/”. Para nuestro ejemplo, usaremos  $\Sigma = \{/, *, a\}$ . Entonces, algunos ejemplos de comentarios en C son:

- /\*aaa\*/
- /\*\*/
- /\*/aaa\*/
- /\*aaa/\*aaa\*/

La Figura 2.1 muestra un ejemplo de autómeta que acepta comentarios en C.

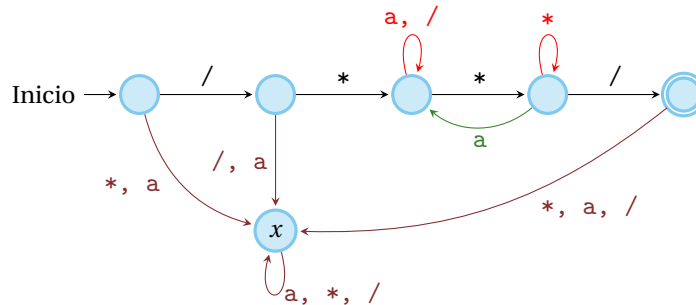


Figura 2.1: Comentarios generales en C. Por otro lado,  $x$  es un *estado muerto*.

**Definición 2.1.** Un *estado muerto* corresponde a aquel estado en que el autómeta no puede salir. Está demás decir, que todo autómeta que entre en un estado muerto rechaza.

### 2.1. Autómatas Finitos Deterministas (DFA)

**Definición 2.2.** Un *autómeta finito determinista* (DFA):

$$M = (Q, \Sigma, \delta, q_0, F) \quad (2.1.1)$$

Consta de:

- $Q$ : conjunto finito de estados.
- $\Sigma$ : alfabeto
- $\delta$ : función de transición,  $\delta : Q \times \Sigma \rightarrow Q$
- $q_0$ : estado inicial, tal que  $q_0 \in Q$ .
- $F$ : estados finales, tales que  $F \subseteq Q$ .

**Observación:** El autómata sólo tiene 1 estado inicial, nunca más.

**Definición 2.3.** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA. Se define la función de transición extendida de  $M$  mediante:

$$\begin{aligned} \hat{\delta}(q, \epsilon) &= q, & ; \text{ para todo } q \in Q \\ \hat{\delta}(q, \alpha x) &= \delta(\hat{\delta}(q, \alpha), x) & ; \text{ para todo } q \in Q, \alpha \in \Sigma^*, x \in \Sigma \end{aligned}$$

**Idea:** Dónde llega  $M$ , partiendo de  $q$  y consumiendo  $\alpha$  es  $\hat{\delta}(q, \alpha)$ .

**Definición 2.4.** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA,  $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$  su función de transición extendida. El lenguaje aceptado por  $M$  es:

$$\mathcal{L}(M) = \{\sigma : \hat{\delta}(q_0, \sigma) \in F\} \quad (2.1.2)$$

En castellano: las palabras en  $\Sigma^*$  que llevan a  $M$  desde  $q_0$  consumiendo  $\sigma$  van a un estado final  $F$ .

**Abuso:** No hay confusión entre  $\delta, \hat{\delta}$

**Definición 2.5.** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA, donde suponemos  $\Sigma \cap Q = \emptyset$ . Una descripción instantánea de  $M$  es una palabra de la forma:

$$\alpha q \beta \quad (2.1.3)$$

con  $\alpha, \beta \in \Sigma^*, q \in Q$ .

La idea de la descripción instantánea (2.1.3) es: “ $M$  leyó  $\alpha$ , está en el estado  $q$  y le falta por leer  $\beta$ ”.

**Definición 2.6.** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA. Definimos la relación de transición de  $M$  entre descripciones instantáneas (IDs) de  $M$  por:

$$\alpha q x \beta \vdash_M \alpha x p \beta \quad ; \text{ para todo } \alpha, \beta \in \Sigma^* \text{ tal que } \delta(q, x) = p \quad (2.1.4)$$

Normalmente escribimos simplemente  $\vdash$  porque siempre hablamos del mismo autómata.

**Definición 2.7.** Sea  $R \subseteq A \times A$  una relación. La clausura transitiva de  $R$ ,  $R^+$  es la misma relación que incluye a  $R$  que es transitiva. La clausura reflexiva y transitiva de  $R$ ,  $R^*$ , es la mínima relación que es reflexiva y transitiva que contiene a  $R$ .

En la práctica:

- $aR^+b$ : De  $a$  llego a  $b$  en 1 o más pasos de  $R$  (con sucesiones se intepreta como el “mayor que”)
- $aR^*b$ : De  $a$  llego a  $b$  en 0 o más pasos de  $R$ . (con sucesiones se intepreta como el “mayor o igual que”)

**Definición 2.8.** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA. El lenguaje aceptado por  $M$  es:

$$\mathcal{L}(M) = \{\sigma : q_0 \sigma \vdash_M^* \sigma q_f \wedge q_f \in F\} \quad (2.1.5)$$

Cabe destacar, que la Definición 2.4 y la Definición 2.8 son equivalentes.



EQVA Convensace de ello...

## 2.2. Autómatas Finitos No Deterministas (NFA)

**Definición 2.9.** Un autómata finito no determinista (NFA):

$$M = (Q, \Sigma, \delta, q_0, F) \quad (2.2.1)$$

consta de:

- $Q$ : conjunto finito de estados.
- $\Sigma$ : alfabeto.
- $\delta$ : función de transición,  $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$ , donde  $2^Q$  es el conjunto de potencia.
- $q_0$ : estado inicial,  $q_0 \in Q$
- $F$ : estados finales,  $F \subseteq Q$

Notar que las movidas no están determinadas. La idea es que  $M$  acepte si hay una secuencia de movidas en las que consume la palabra y termina en un estado final.

**Definición 2.10.** Definimos ID para NFA igual que para DFA.

**Definición 2.11.** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un NFA. Definimos la relación de transición de  $M$  entre IDs de  $M$  mediante:

- I.  $\alpha q \beta \vdash_M \alpha p \beta$ , si  $p \in \delta(q, \epsilon)$
- II.  $\alpha q x \beta \vdash_M \alpha p \beta$  si  $p \in \delta(q, x)$

Donde  $\alpha, \beta \in \Sigma^*$ ,  $p, q \in Q$ ,  $x \in \Sigma$ .

**Definición 2.12.** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un NFA. El lenguaje aceptado por  $M$  es:

$$\mathcal{L}(M) = \{\sigma : q_0 \sigma \vdash_M^* \sigma q_f \wedge q_f \in F\} \quad (2.2.2)$$

Nótese que las definiciones 2.12 y 2.8 son exactamente iguales.

**Ejemplo 2.2** (Ejemplo NFA). Palabras sobre  $\{a, b, c\}$  comiencen en una secuencia de  $a$ 's y  $b$ 's que terminen en  $abc$  (que es lo mismo que la expresión regular  $(a|b)^* abc$ ):

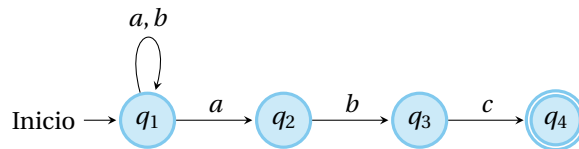
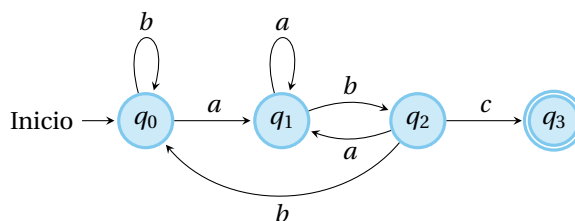


Figura 2.2: El NFA tiene la opción de elegir ir al estado  $q_1$  o al  $q_2$  consumiendo  $a$  cuando se encuentra en  $q_1$ . Esta capacidad del NFA de poder “elegir” o “adivinar” siempre el camino correcto se le llama no-determinismo.

En contraste con el NFA de la Figura 2.2, un DFA para la expresión regular  $(a|b)^* abc$  sería:

Como podemos observar, la construcción de un DFA es más complicada que la de un NFA, dado que el no-determinismo nos permite hacer cosas que el DFA no permite, como las transiciones con  $\epsilon$  por dar un ejemplo.

Figura 2.3: DFA de la expresión regular  $(a|b)^* abc$ .

## 2.3. Relaciones entre DFA, NFA y Expresiones Regulares

### 2.3.1. Obtención de un NFA a partir de una expresión regular

DFA es caso particular de NFA:

- $\delta(q, \epsilon) = \emptyset$  para todo  $q \in Q$
- $|\delta(q, x)| = 1$  para todo  $q \in Q, x \in \Sigma$
- Por lo tanto, aceptado por DFA  $\subseteq$  aceptado por NFA

R.E.  $\rightarrow$  NFA:

- Dada una R.E. sobre  $\Sigma$ , construimos un NFA que acepta el lenguaje que denota.

**Teorema 2.1.** Sea  $R$  una expresión regular sobre  $\Sigma$ . Construimos un NFA que acepte  $\mathcal{L}(R)$ .

*Demostración.* De la definición de expresiones regulares (R.E.) podemos decir lo siguiente:

- i. Equivalencia entre R.E. y NFA de lenguaje vacío  $\emptyset$ :



- ii. Equivalencia entre R.E. y NFA de palabra vacía  $\epsilon$ :



- iii. Equivalencia entre R.E. y NFA de un símbolo  $a$ , tal que  $a \in \Sigma$ :



Notar que todos los NFA contruidos tienen un estado final, y una vez que se “entra” no hay forma de volver atrás; y una vez “sale” (llega al final) no hay manera de devolverse.

Ahora supongamos que tenemos dos expresiones regulares  $R$  y  $S$  tal que estas tienen sus respectivos NFA's indicados a continuación:

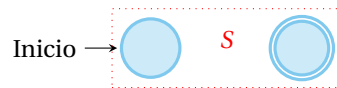
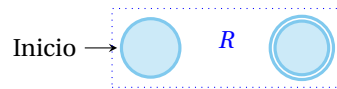
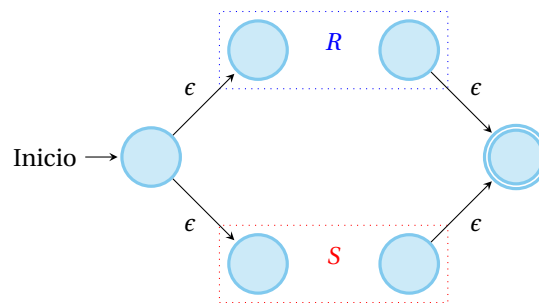


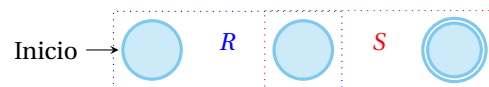
Figura 2.4: Los rectángulos que encierran los estados con líneas punteadas representan el NFA de la expresión regular correspondiente.

Luego, podemos decir:

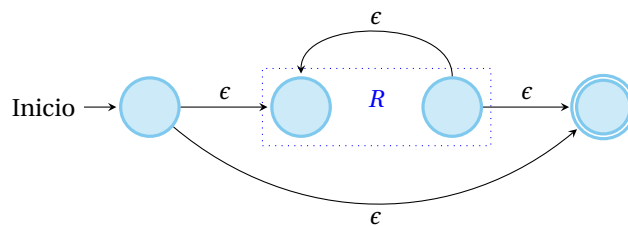
- Operación OR entre dos expresiones regulares:  $(R) \mid (S)$ :



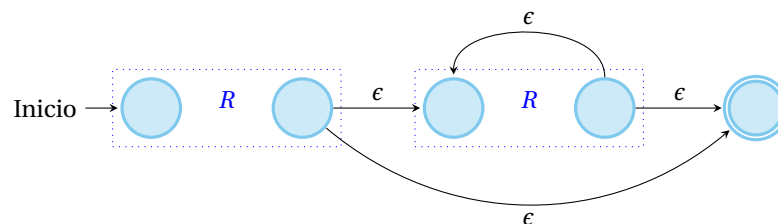
- Concatenación entre dos expresiones regulares:  $(R) \cdot (S)$ :



- Estrella Kleene de una expresión regular:  $(R)^*$ :



- Kleene Plus de una expresión regular:  $(R)^+ = (R) \cdot (R)^*$ :



### 2.3.2. Obtención de un DFA a partir de un NFA

La Figura 2.5 describe lo que llevamos hasta el momento.

R.E.  $\longrightarrow$  NFA

DFA

Figura 2.5: Gracias al teorema 2.1, podemos obtener un NFA a partir de una expresión regular.

**Idea:** Los estados del DFA registran el conjunto de estados en que puede estar el NFA.

Problemas del NFA:

- Transiciones  $\epsilon$ .

$\rightarrow \epsilon$ -closure( $A$ ) = conjunto de estados que pueden alcanzarse mediante transiciones  $\epsilon$ .

Para calcularlo:

- $\mathcal{A} \subseteq \epsilon$ -closure( $\mathcal{A}$ )
- Si  $q \in \epsilon$ -closure( $\mathcal{A}$ ), y  $p \in \delta(q, \epsilon)$ , entonces  $p \in \epsilon$ -closure( $\mathcal{A}$ ). La idea es repetir los anteriores hasta que no hayan cambios.
- El conjunto  $2^Q$  es gigante  $\Rightarrow$  construiremos sólo los conjuntos alcanzables desde el estado inicial.

#### 2.3.2.1. Algoritmo

Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un NFA. Definimos el DFA  $M' = (2^Q, \Sigma, \delta', q'_0, F')$  mediante lo siguiente:

- $q_0 = \epsilon$ -closure( $\{q_0\}$ )
- $F' = \{\mathcal{A} \subseteq Q : \mathcal{A} \cap F \neq \emptyset\}$
- $\delta'(\mathcal{A}, x) = \bigcup_{q \in \mathcal{A}} \epsilon$ -closure( $\delta(q, x)$ ) ; para todo  $\mathcal{A} \subseteq Q, x \in \Sigma$

Entonces  $\mathcal{L}(M) = \mathcal{L}(M')$

**Ejemplo 2.3** (Convertir NFA a DFA). Consideremos palabras sobre  $\Sigma = \{a, b, c\}$  descritos por:

$$(a|b)^* abc \tag{2.3.1}$$

Lo primero que tenemos que hacer, es obtener el NFA de la expresión regular (2.3.1). Para ello, le aplicamos el Teorema 2.1 conllevando al resultado dado por la Figura 2.6.

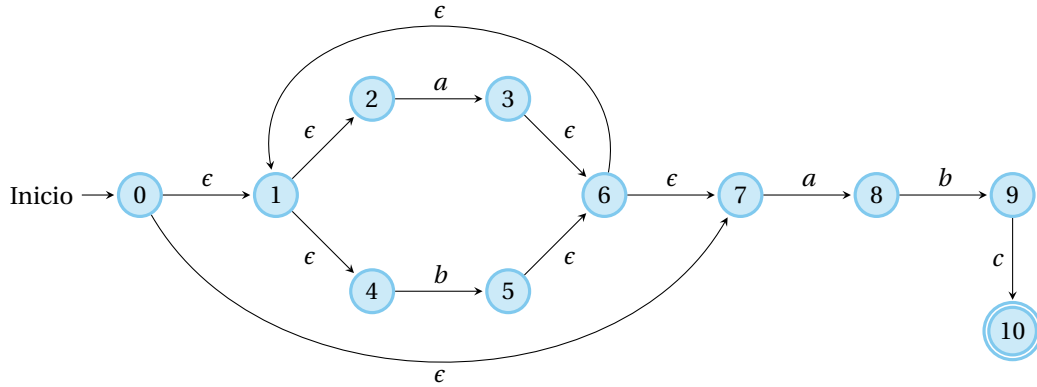


Figura 2.6: NFA de la expresión regular  $(a|b)^*abc$  tras aplicar el método de Thompson.

Comenzamos contruyendo una tabla de la forma:

Conjunto	$a$	$b$	$c$	Tipo

Cuadro 2.1: Tabla inicial de conversión NFA a DFA. La completaremos con el tiempo.

Comenzamos aplicando la operación  $\epsilon$  – closure() al estado inicial. Sólo para este estado lo haremos paso por paso:

- De estado 0 podemos ir a los estados 0, 1 y 7. Por lo tanto, hasta el momento  $\epsilon$  – closure({0}) viene dado por:

$$\epsilon - \text{closure}(\{0\}) = \{0, 1, 7, \dots \quad (2.3.2)$$

Nótese que el estado 0 también va como resultado de la operación  $\epsilon$  – closure(). Esto se debe a que podemos acceder a través del mismo consumiendo  $\epsilon$  a través de un loop, algo totalmente válido para un NFA.

Antes de continuar, le agregamos una marca al estado 0 para recordarnos que ya encontramos los estados a los cuáles podemos acceder a través de él consumiendo  $\epsilon$ :

$$\epsilon - \text{closure}(\{0\}) = \{\overset{\checkmark}{0}, 1, 7, \dots \quad (2.3.3)$$

- Recuerde que  $\epsilon$  – closure() es una operación recursiva, por lo tanto tendremos que ver a qué estados podemos acceder consumiendo  $\epsilon$  de los que obtuvimos con 0.

Continuamos con el estado 1, y a través de él consumiendo  $\epsilon$  podemos acceder a los estados 2 y 4. Los agregamos al conjunto de salida de (2.3.3):

$$\epsilon - \text{closure}(\{0\}) = \{\overset{\checkmark}{0}, 1, 7, 2, 4, \dots$$

Luego, le colocamos una marca al estado 1 para recordarnos que ya vimos a qué estados podemos ir consumiendo  $\epsilon$ :

$$\epsilon - \text{closure}(\{0\}) = \{\overset{\checkmark}{0}, \overset{\checkmark}{1}, 7, 2, 4, \dots \quad (2.3.4)$$

- Continuamos con el estado 7. A través del estado 7 consumiendo  $\epsilon$  sólo podemos ir al mismo 7. Por lo tanto, no agregamos nada al conjunto de salida de (2.3.4) y colocamos una marca en el estado 7:

$$\epsilon - \text{closure}(\{0\}) = \{\overset{\checkmark}{0}, \overset{\checkmark}{1}, \overset{\checkmark}{7}, 2, 4, \dots \quad (2.3.5)$$

- Continuamos con el estado 2. Ocurre lo mismo que con 7, consumiendo  $\epsilon$  sólo podemos ir a 2. Por lo tanto el conjunto de salida de (2.3.5) queda intacto. Antes de pasar al siguiente le colocamos una marca al estado 2:

$$\epsilon - \text{closure}(\{0\}) = \{\overset{\checkmark}{0}, \overset{\checkmark}{1}, \overset{\checkmark}{7}, \overset{\checkmark}{2}, 4 \dots \quad (2.3.6)$$

- Seguimos con el estado 4. De la misma forma que los estados 2 y 7, sólo podemos acceder al estado 4 consumiendo  $\epsilon$ . En consecuencia, el conjunto de salida de (2.3.6) queda intacto. En este momento, agregamos una marca a dicho estado:

$$\epsilon - \text{closure}(\{0\}) = \{\overset{\checkmark}{0}, \overset{\checkmark}{1}, \overset{\checkmark}{7}, \overset{\checkmark}{2}, \overset{\checkmark}{4} \dots \quad (2.3.7)$$

- Ahora que tenemos con una marca todos los estados del conjunto de salida de  $\epsilon - \text{closure}()$ , vemos que el resultado de  $\epsilon - \text{closure}(\{0\})$  es el mismo (2.3.7). Los reordenamos para mantener el orden:

$$\epsilon - \text{closure}(\{0\}) = \{0, 1, 2, 4, 7\} \quad (2.3.8)$$

El resultado de (2.3.8) no se encuentra en nuestra tabla, así que lo agregamos y le damos un nombre...  $A$ :

	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$				

Luego, para cada estado de  $A$  vemos a cuáles podemos acceder consumiendo  $a$ ,  $b$  y  $c$ . A los resultados arrojados le aplicamos  $\epsilon - \text{closure}()$ .

**Convención:** Usaremos la notación  $q \rightarrow \sigma$ , con  $q \in 2^Q$  y  $\sigma \in \Sigma$  para mostrar a qué estados podemos ir desde  $q$  consumiendo  $\sigma$  (el resultado es un conjunto):

Comenzamos con  $A \rightarrow a$  (¿a qué estados podemos ir desde el conjunto de estados  $A$  consumiendo  $a$ ). Recordemos que  $A = \{0, 1, 2, 4, 7\}$ , entonces:

- De los estados: 0, 1, 4 no vamos a ninguna parte consumiendo  $a$ .
- Del estado 2 vamos al estado 3 consumiendo  $a$ . Agregamos este estado al conjunto de salida de  $A \rightarrow a$ :

$$A \rightarrow a = \{3, \dots$$

- Del estado 7 vamos al estado 8 consumiendo  $a$ . Agregamos este estado al conjunto de salida de  $A \rightarrow a$ :

$$A \rightarrow a = \{3, 8, \dots$$

- Como ya vimos todos los estados de  $A$  a los cuales podemos ir consumiendo  $a$ , vemos que:

$$A \rightarrow a = \{3, 8\} \quad (2.3.9)$$

Inmediatamente, aplicamos la operación  $\epsilon - \text{closure}()$  a (2.3.9):

$$\epsilon - \text{closure}(\{3, 8\}) = \{1, 2, 3, 4, 6, 7, 8\} \quad (2.3.10)$$

El conjunto  $\{1, 2, 3, 4, 6, 7, 8\}$  no se encuentra en la tabla, por ende se agrega y se bautiza:

	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$	$B$			
$B$	$\{1, 2, 3, 4, 6, 7, 8\}$				

Continuamos donde quedamos:

$$A \rightarrow b = \{5\}$$

$$\epsilon - \text{closure}(\{5\}) = \{1, 2, 4, 5, 6, 7\} \quad (2.3.11)$$

El resultado (2.3.11) no se encuentra en la tabla, así que lo agregamos y le damos un nombre:

	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$	$B$	$C$		
$B$	$\{1, 2, 3, 4, 6, 7, 8\}$				
$C$	$\{1, 2, 4, 5, 6, 7\}$				

De inmediato, seguimos con  $A \rightarrow c$ :

$$A \rightarrow c = \emptyset$$

$$\epsilon - \text{closure}(\emptyset) = \emptyset$$

El conjunto vacío no está en nuestra tabla, así que lo agregamos:

	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$	$B$	$C$	$D$	
$B$	$\{1, 2, 3, 4, 6, 7, 8\}$				
$C$	$\{1, 2, 4, 5, 6, 7\}$				
$D$	$\emptyset$				

Rápidamente, vemos que del conjunto vacío sin importar lo que consumamos no llegaremos a ningún estado, es decir, uno vacío. Entonces:

	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$	$B$	$C$	$D$	
$B$	$\{1, 2, 3, 4, 6, 7, 8\}$				
$C$	$\{1, 2, 4, 5, 6, 7\}$				
$D$	$\emptyset$	$D$	$D$	$D$	

Realizamos el mismo procedimiento para  $B$ :

$$B \rightarrow a = \{3, 8\}$$

Ya hemos aplicado previamente  $\epsilon - \text{closure}()$  a  $\{3, 8\}$  (ecuación (2.3.10)). Por lo tanto, es obvio que del conjunto  $B$  consumiendo  $a$  llegamos al mismo  $B$ :

Continuamos:

$$B \rightarrow b = \{5, 9\}$$

$$\epsilon - \text{closure}(\{5, 9\}) = \{1, 2, 4, 5, 6, 7, 9\}$$

	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$	$B$	$C$	$D$	
$B$	$\{1, 2, 3, 4, 6, 7, 8\}$	$B$			
$C$	$\{1, 2, 4, 5, 6, 7\}$				
$D$	$\emptyset$	$D$	$D$	$D$	

$\{1, 2, 4, 5, 6, 7, 9\}$  no está en la tabla. Así que lo agregamos con nombre incluido:

	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$	$B$	$C$	$D$	
$B$	$\{1, 2, 3, 4, 6, 7, 8\}$	$B$	$E$		
$C$	$\{1, 2, 4, 5, 6, 7\}$				
$D$	$\emptyset$	$D$	$D$	$D$	
$E$	$\{1, 2, 4, 5, 6, 7, 9\}$				

Retomamos:

$$B \rightarrow c = \emptyset$$

Ya hemos obtenido el conjunto vacío antes, por lo tanto:

	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$	$B$	$C$	$D$	
$B$	$\{1, 2, 3, 4, 6, 7, 8\}$	$B$	$E$	$D$	
$C$	$\{1, 2, 4, 5, 6, 7\}$				
$D$	$\emptyset$	$D$	$D$	$D$	
$E$	$\{1, 2, 4, 5, 6, 7, 9\}$				

Pasamos al conjunto de estados  $C$  y hacemos lo de siempre:

$$C \rightarrow a = \{3, 8\} \rightsquigarrow B$$

$$C \rightarrow b = \{5\} \rightsquigarrow C$$

$$C \rightarrow c = \emptyset \rightsquigarrow D$$

Agregamos la información anterior a la tabla:

	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$	$B$	$C$	$D$	
$B$	$\{1, 2, 3, 4, 6, 7, 8\}$	$B$	$E$	$D$	
$C$	$\{1, 2, 4, 5, 6, 7\}$	$B$	$C$	$D$	
$D$	$\emptyset$	$D$	$D$	$D$	
$E$	$\{1, 2, 4, 5, 6, 7, 9\}$				

Continuamos el supuesto último conjunto de datos  $E$ :

$$E \rightarrow a = \{3, 8\} \rightsquigarrow B$$

$$E \rightarrow b = \{5\} \rightsquigarrow C$$

$$E \rightarrow c = \{10\}$$

$$\epsilon\text{-closure}(\{10\}) = \{10\}$$

El conjunto arrojado por  $\epsilon\text{-closure}(\{10\})$  no está en la tabla, así que lo agregamos con un nombre único:



	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$	$B$	$C$	$D$	
$B$	$\{1, 2, 3, 4, 6, 7, 8\}$	$B$	$E$	$D$	
$C$	$\{1, 2, 4, 5, 6, 7\}$	$B$	$C$	$D$	
$D$	$\emptyset$	$D$	$D$	$D$	
$E$	$\{1, 2, 4, 5, 6, 7, 9\}$	$B$	$C$	$F$	
$F$	$\{10\}$				

Nuevamente, para cada estado en  $F$  vemos hacia dónde vamos consumiendo  $a$ ,  $b$  y  $c$ :

$$F \rightarrow a = \emptyset \rightsquigarrow D$$

$$F \rightarrow b = \emptyset \rightsquigarrow D$$

$$F \rightarrow c = \emptyset \rightsquigarrow D$$

No tenemos más estados nuevos, así que vamos definiendo el “Tipo” de cada uno de los conjuntos anteriores:

- El **estado inicial** está dado por  $\epsilon$  – closure  $(\{0\})$ , que en este caso es  $A$ .
- Por otro lado, todos aquellos que contengan al estado final de nuestro NFA inicial (Figura 2.6), serán **estados finales** del DFA resultante. Sólo el estado  $F$  contiene a 10, por lo tanto es final.

Finalmente, se tiene que la tabla de transición de estados  $\delta$  del DFA resultante es:

	Conjunto	$a$	$b$	$c$	Tipo
$A$	$\{0, 1, 2, 4, 7\}$	$B$	$C$	$D$	Inicial
$B$	$\{1, 2, 3, 4, 6, 7, 8\}$	$B$	$E$	$D$	
$C$	$\{1, 2, 4, 5, 6, 7\}$	$B$	$C$	$D$	
$D$	$\emptyset$	$D$	$D$	$D$	
$E$	$\{1, 2, 4, 5, 6, 7, 9\}$	$B$	$C$	$F$	
$F$	$\{10\}$	$D$	$D$	$D$	Final

Cuadro 2.2: Tabla de transición del DFA resultante

A través de la tabla de transición definida en el Cuadro 2.2, vemos que el DFA es:

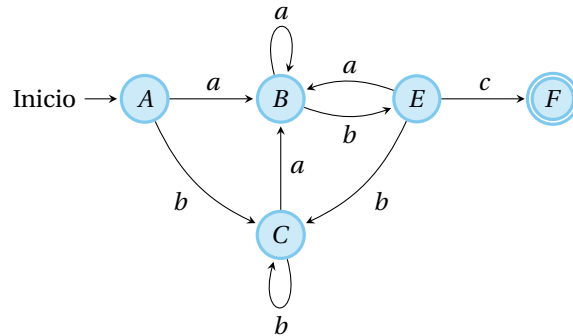


Figura 2.7: Un DFA para la expresión regular  $(a|b)^* abc$

El DFA de la Figura 2.7 es sólo un **candidato**. Existen otros NFA que representan la misma expresión regular, como el que muestra la Figura 2.8.

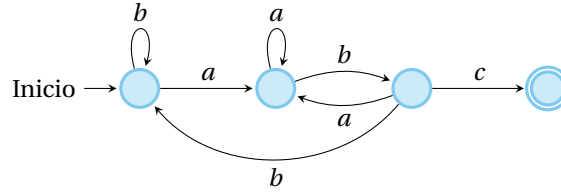


Figura 2.8: Omitiendo el estado muerto del DFA de la Figura 2.7.

Con ello, completamos el paso NFA  $\rightarrow$  DFA (Figura 2.9).

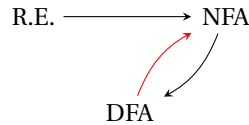


Figura 2.9: La flecha roja representa que DFA es un caso particular de NFA.

### 2.3.3. Obtención de una Expresión Regular a partir de un DFA

Este truco se usa en variados algoritmos que procesan grafos.

**Idea:** Dado  $M = (Q, \Sigma, \delta, q_0, F)$ , numeramos (¡arbitrariamente!) los estados  $Q = \{1, 2, 3, \dots, n\}$ .

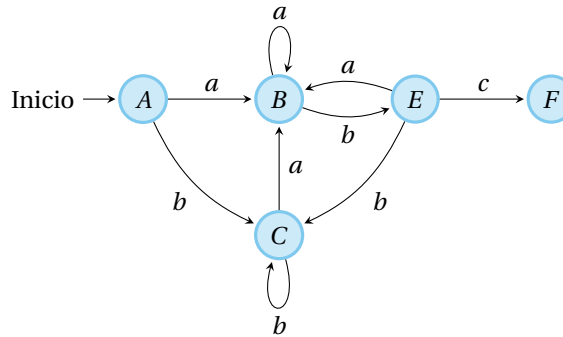
Definimos:

- $R_{ij}^{(k)}$ : Expresión regular denota las palabras que llevan a  $M$  del estado  $i$  al estado  $j$ , pasando sólo por estados  $\{1, 2, \dots, k\}$  entremedio.

Los  $R_{ij}^{(n)}$  son las palabras que llevan a  $M$  del estado  $i$  al  $j$  sin restricciones.

- $R_{q_0, q_f}^{(n)}$ : llevan a  $M$  del estado inicial  $q_0$  al final  $q_f \rightarrow$  | de todos estos denota el lenguaje que  $M$  acepta.

*Demostración (inducción).* Para un ejemplo de la demostración, considere el DFA:



**Base**  $R_{ij}^{(0)}$ : Ir de  $i$  a  $j$  directamente. En el ejemplo:

$$R_{AB}^{(0)} = a, \quad R_{AE}^{(0)} = \emptyset, \quad R_{EE}^{(0)} = \epsilon, \quad R_{BB}^{(0)} = \epsilon \mid a \quad \dots$$

**Inducción** Dados los  $R_{ij}^{(k)}$ , calculamos  $R_{ij}^{(k+1)}$ . Opciones:

- De  $i$  a  $j$  ( $R_{ij}^{(k)}$ ) pasando sólo por  $\{1, \dots, k\}$ .
- De  $i$  a  $j$ , pasando al menos una vez por  $k+1$  (Figura 2.10).

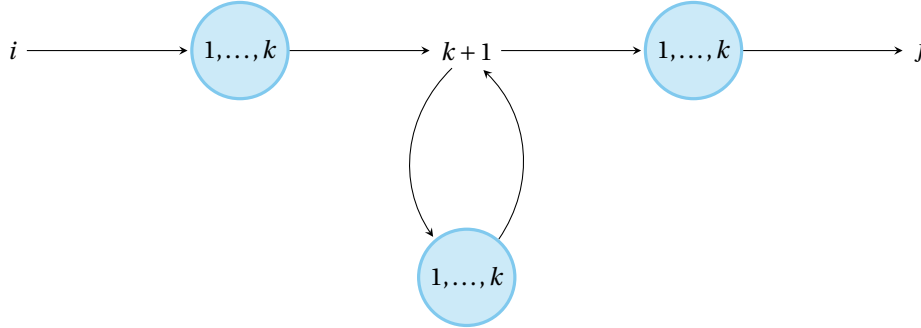


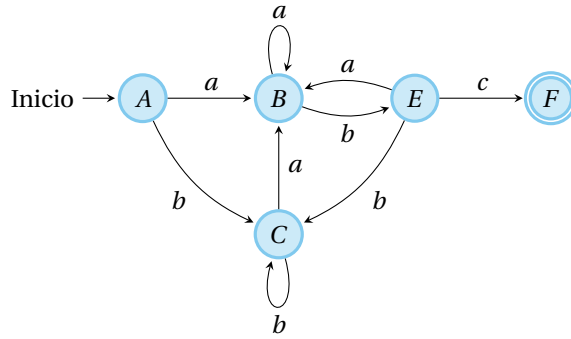
Figura 2.10: Los nodos encerrados en círculos representan el conjunto de estados  $\{1, \dots, k\}$

Es decir:

$$R_{ij}^{(k+1)} = R_{ij}^{(k)} \mid R_{i,k+1}^{(k)} \left( R_{k+1,k+1}^{(k)} \right)^* R_{k+1,j}^{(k)} \quad (2.3.12)$$

□

**Otra Idea** Sistema de ecuaciones:



Sea  $L_{ij}$  los lenguajes de palabras que llevan de  $i$  a  $j$ . Por ejemplo, ir de  $A$  a  $C$ :

$$L_{AA} = \epsilon$$

$$L_{AC} = bL_{CC}$$

$$L_{AB} = aL_{BB}$$

$\vdots$

Otra forma es cómo llegamos a un estado (recuerde partir del inicial):

$$L_A = \epsilon \quad (\text{de } A \text{ a } A)$$

$$L_B = L_A a \mid L_B a \mid L_E a \mid L_C a$$

$\vdots$

Para resolver algunas ecuaciones utilice:

$$X = A \cdot X \mid B \quad (2.3.13)$$

Que vimos hace algun tiempo. Recuerde que la respuesta de lo anterior es:

$$X = A^* B$$

Con ello, tenemos las herramientas necesarias para resolver las 25 ecuaciones.  $\square$

Completamos el ciclo:

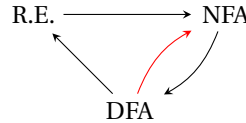


Figura 2.11: La flecha roja representa un caso particular de los DFA.

## 2.4. DFA Minimo

**Definición 2.13** (DFA mínimo). Corresponde un DFA con mínimo número de estados.

**Teorema 2.2** (Myhill - Nerode). Si  $M = (Q, \Sigma, \delta, q_0, F)$  es un DFA y  $M' = (Q', \Sigma, \delta', q'_0, F')$  es el DFA mínimo tal que:

$$\mathcal{L}(M) = \mathcal{L}(M'), \quad (2.4.1)$$

entonces los estados de  $Q'$  corresponden a una partición de  $Q$  (cada estado  $q' \in Q'$  corresponde a un conjunto de estados de  $Q$ ).

Decimos que en el DFA  $M = (Q, \Sigma, \delta, q_0, F)$  la palabra  $\sigma \in \Sigma^*$  distingue entre  $p, q \in Q$  si  $\delta(p, \sigma) \in F \Leftrightarrow \delta(q, \sigma) \notin F$ . Qué es lo mismo que:

$$(\delta(p, \sigma) \in F \wedge \delta(q, \sigma) \notin F) \vee (\delta(p, \sigma) \notin F \wedge \delta(q, \sigma) \in F)$$

**Cuidado:** Distinguir entre  $p$  y  $q$  no nos dice el camino tomado por el DFA hasta llegar a  $q$ .

**Idea General:** Queremos construir una partición de  $Q$  tal que estados en el mismo grupo no se distinguen. Esto partiendo con un grupo, e ir viendo si alguno de los grupos debe dividirse si con un símbolo hay estados que llevan a grupos diferentes.

### 2.4.0.1. Algoritmo para encontrar el DFA mínimo

1. Tome  $Q$  y divídalo en dos conjuntos (grupos): estados finales y estados no finales.
2. Realice una tabla de transición de estados. El resultado de salida será el conjunto de estados al que se accede consumiendo  $\sigma \in \Sigma$  en lugar el estado  $q \in Q$ .
3. Compare las filas de cada grupo en la tabla. Si en un grupo existen filas diferentes, divídalo en tantos grupos sea posible como filas diferentes existan. y vuelva al paso 2.

**Nota:** Agrupe las filas por grupo de los subconjuntos resultantes.

4. Si para cada grupo existente, las filas que lo componen son iguales, encontró el DFA mínimo.

**Ejemplo 2.4.** Consideremos como ejemplo el DFA de la Figura 2.12.

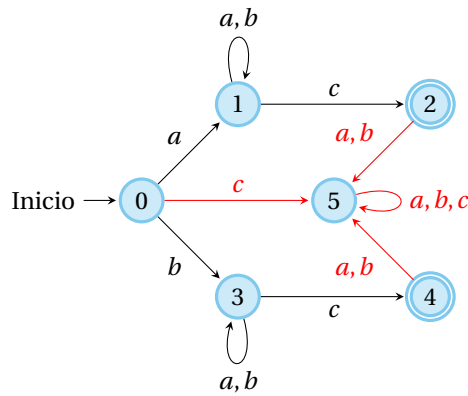


Figura 2.12: DFA de ejemplo. Le llamaremos  $M$ .

Creamos una tabla de transición inicial:

	$a$	$b$	$c$	Tipo
0	1	3	5	Inicial
1	1	1	1	
2	5	5	5	Final
3	3	3	4	
4	5	5	5	Final
5	5	5	5	

Cuadro 2.3: Tabla de transición de estados de  $M$ .

Creamos dos grupos, subconjunto tendrá los estados no finales y el otro los finales:

$$A = \{0, 1, 3, 5\}$$

$$B = \{2, 4\}$$

Luego, agrupamos las filas del Cuadro 2.3 según el grupo en el que se encuentren:

	$a$	$b$	$c$
$A$	0	$A$	$A$
	1	$A$	$B$
	3	$A$	$B$
	5	$A$	$A$
$B$	2	$A$	$A$
	4	$A$	$A$

De la tabla anterior, vemos que las filas que tienen el estado 1 y 3 (combinación  $AAB$ ) son diferentes de las 0 y 5 (combinación  $AAA$ ). Por lo tanto, las dividimos en dos grupos donde:

$$A = \{0, 5\}$$

$$B = \{1, 3\}$$

$$C = \{2, 4\}$$

Luego, la tabla de transición de la segunda división es:

		<i>a</i>	<i>b</i>	<i>c</i>
<i>A</i>	0	<i>B</i>	<i>B</i>	<i>A</i>
	5	<i>A</i>	<i>A</i>	<i>A</i>
<i>B</i>	1	<i>B</i>	<i>B</i>	<i>C</i>
	3	<i>B</i>	<i>B</i>	<i>C</i>
<i>C</i>	2	<i>A</i>	<i>A</i>	<i>A</i>
	4	<i>A</i>	<i>A</i>	<i>A</i>

Como puede ver en el grupo *A* de la tabla anterior, sus filas difieren, en consecuencia, creamos nuevos grupos dados por:

$$A = \{0\}$$

$$B = \{5\}$$

$$C = \{1, 3\}$$

$$D = \{2, 4\}$$

Luego, la tabla de transición de estados para la tercera división es:

		<i>a</i>	<i>b</i>	<i>c</i>
<i>A</i>	0	<i>C</i>	<i>C</i>	<i>B</i>
<i>B</i>	5	<i>B</i>	<i>B</i>	<i>B</i>
<i>C</i>	1	<i>C</i>	<i>C</i>	<i>D</i>
	3	<i>C</i>	<i>C</i>	<i>D</i>
<i>D</i>	2	<i>B</i>	<i>B</i>	<i>B</i>
	4	<i>B</i>	<i>B</i>	<i>B</i>

Como ya no tenemos filas diferentes dentro de un mismo grupo, marcamos los estados no finales y finales bajo el mismo criterio de siempre. La Figura 2.13 muestra con dibujitos el DFA mínimo que

		<i>a</i>	<i>b</i>	<i>c</i>	Tipo
<i>A</i>	0	<i>C</i>	<i>C</i>	<i>B</i>	Inicial
<i>B</i>	5	<i>B</i>	<i>B</i>	<i>B</i>	
<i>C</i>	1	<i>C</i>	<i>C</i>	<i>D</i>	
	3	<i>C</i>	<i>C</i>	<i>D</i>	
<i>D</i>	2	<i>B</i>	<i>B</i>	<i>B</i>	Final
	4	<i>B</i>	<i>B</i>	<i>B</i>	

Cuadro 2.4: Tabla de transición del DFA mínimo.

resulta del Cuadro 2.4.

**Ejemplo 2.5.** Queremos encontrar el DFA mínimo  $M = (Q, \Sigma, \delta, q_0, F)$  dado por la Figura 2.14. Para ello, comenzamos creando los dos conjuntos: los que no son finales y los que sí son finales:

$$A = \{1, 2, 3, 4, 6\}$$

$$B = \{5\}$$

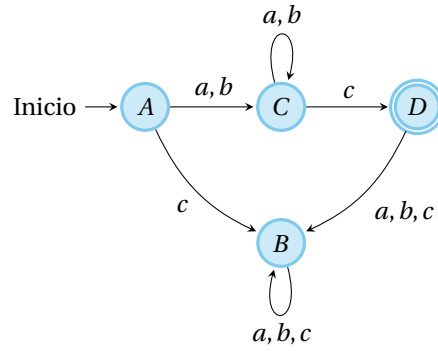


Figura 2.13: DFA mínimo de  $M$ .

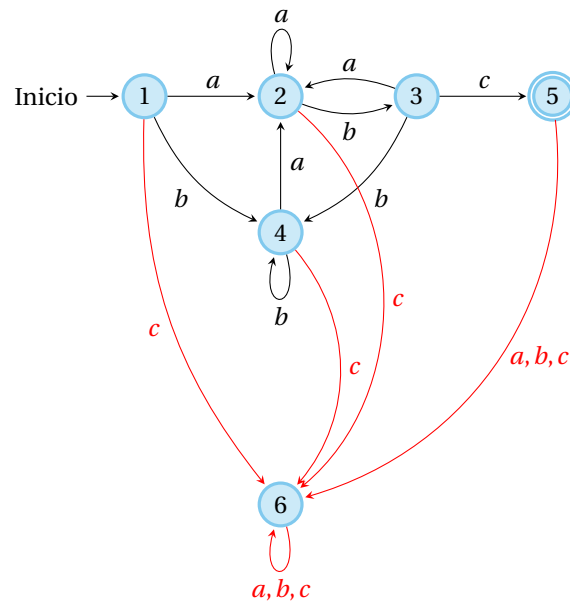


Figura 2.14: DFA candidato resultante del ejemplo 2.3 (Figura 2.7).  
Nótese que le agregamos un estado muerto

Luego, la tabla de transición para la primera división es:

		$a$	$b$	$c$
	1	$A$	$A$	$A$
	2	$A$	$A$	$A$
$A$	3	$A$	$A$	$B$
	4	$A$	$A$	$A$
	6	$A$	$A$	$A$
$B$	5			

Cuadro 2.5: No hay posibilidad de división del grupo  $B$ , así que dejaremos su fila en blanco por el momento. Haremos lo mismo con los grupos posteriores.

En el Cuadro 2.5, la única fila que difiere de las demás en el conjunto  $A$  es el estado 3. En conse-

cuencia, separamos los nuevos grupos son:

$$A = \{1, 2, 4, 6\}$$

$$B = \{3\}$$

$$C = \{5\}$$

La tabla de transición de la segunda división es:

		<i>a</i>	<i>b</i>	<i>c</i>
<i>A</i>	1	<i>A</i>	<i>A</i>	<i>A</i>
	2	<i>A</i>	<i>B</i>	<i>A</i>
	4	<i>A</i>	<i>A</i>	<i>A</i>
	6	<i>A</i>	<i>A</i>	<i>A</i>
<i>B</i>	3			
<i>C</i>	5			

Tercera división:

$$A = \{1, 4, 6\}$$

$$B = \{2\}$$

$$C = \{3\}$$

$$D = \{5\}$$

Tabla de transición de la tercera división:

		<i>a</i>	<i>b</i>	<i>c</i>
<i>A</i>	1	<i>B</i>	<i>A</i>	<i>A</i>
	4	<i>B</i>	<i>A</i>	<i>A</i>
	6	<i>A</i>	<i>A</i>	<i>A</i>
<i>B</i>	2			
<i>C</i>	3			
<i>D</i>	5			

Cuarta división:

$$A = \{1, 4\}$$

$$B = \{6\}$$

$$C = \{2\}$$

$$D = \{3\}$$

$$E = \{5\}$$

El Cuadro 2.6 muestra la tabla de transición de la cuarta división. Como no hay filas que difieran dentro de un mismo conjunto de estados, se tiene que el DFA *M* es el grafo dirigido de la Figura 2.15.



		<i>a</i>	<i>b</i>	<i>c</i>	Tipo
<i>A</i>	1	<i>B</i>	<i>A</i>	<i>E</i>	Inicial
	4	<i>B</i>	<i>A</i>	<i>E</i>	
<i>B</i>	6	<i>B</i>	<i>C</i>	<i>E</i>	
<i>C</i>	2	<i>B</i>	<i>A</i>	<i>D</i>	
<i>D</i>	3	<i>E</i>	<i>E</i>	<i>E</i>	Final
<i>E</i>	5	<i>E</i>	<i>E</i>	<i>E</i>	Muerto

Cuadro 2.6: Tabla de transición resultante de la cuarta división.

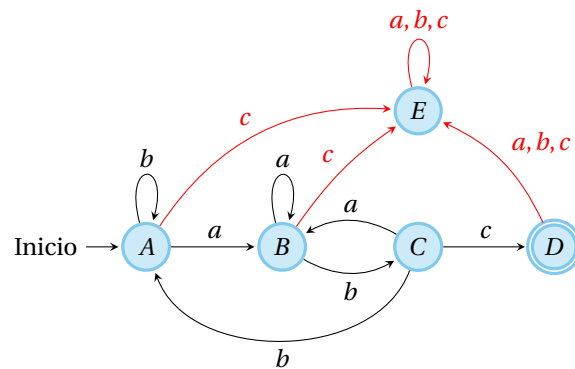


Figura 2.15: DFA mínimo de  $M$ , correspondiente a la expresión regular  $(a|b)^* abc$



## Capítulo 3

# Propiedades de los Lenguajes Regulares

¿Hay lenguajes no regulares? ¿Cómo demostramos que un lenguaje no es regular?. Hasta el momento sólo sabemos que para que un lenguaje sea regular, debe existir un DFA o NFA sea generado a partir de la expresión regular que genera dicho lenguaje. En este capítulo, veremos otras formas de demostrar que un lenguaje es regular, ya sea por el Lema de Bombeo, o las propiedades de clausura para lenguajes regulares.

### 3.1. Lema de Bombeo

---

**Teorema 3.1** (Lema de Bombeo). Si  $L$  es un lenguaje regular, hay una constante  $N > 0$  tal que si  $\sigma \in L$ ,  $|\sigma| \geq N$  podemos escribir:

$$\sigma = \alpha\beta\gamma$$

con  $|\alpha\beta| \leq N$ ,  $\beta \neq \epsilon$  tal que para todo  $k \geq 0$ :

$$\alpha\beta^k\gamma \in L$$

*Demostración.* Sea  $L = \mathcal{L}(M)$  donde  $M = (Q, \Sigma, \delta, q_0, F)$  es un DFA. Sea  $N = |Q|$ , y sea  $\sigma \in L$  con  $|\sigma| \geq N$ , y consideremos los estados por los que pasa  $M$  al procesar  $\sigma = a_1 a_2 \cdots a_n$

$$\begin{array}{cccccc} & a_1 & & a_2 & & \cdots & & a_n \\ q_0 & & q_1 & & q_2 & & q_{n-1} & & q_n \end{array}$$

Esta es una lista de  $n + 1 > |Q|$  estados, por el principio del palomar al menos uno se repite.

Sea  $q_i = q_j$  el primer índice para el que un estado se repite,  $i < j$ . Es claro que  $j \leq N$ , porque:

$$\left| \begin{array}{cccccc} & a_1 & & a_2 & & \cdots & & a_N \\ q_0 & & q_1 & & q_2 & & q_{N-1} & & q_N \end{array} \right| \cdots \begin{array}{cc} & a_n \\ q_{N-1} & & q_N \end{array}$$

La repetición ocurre en este rango

Vale decir:

Sea  $a_1 \cdots a_i = \alpha$ ,  $a_{i+1} \cdots a_j = \beta$ ,  $a_{j+1} \cdots a_n = \gamma$ .

$$\begin{array}{ccccccc}
 & a_1 & \cdots & a_i & \cdots & a_j & \cdots & a_N & \cdots & a_n \\
 q_0 & & & & q_i & & q_j = q_i & & & q_n \in F
 \end{array}$$

Sabemos:

$$|\alpha\beta| \leq N, \quad \beta \neq \epsilon$$

$\delta(q_i, \beta) = q_i$ ,  $\delta(q_0, \alpha) = q_i$ ,  $\delta(q_i, \gamma) \in F$ . O sea,  $\delta(q_0, \alpha\beta^k\gamma) \in F$ . Vale decir,  $\alpha\beta^k\gamma \in L$ , para todo  $k \geq 0$ .  $\square$

**Observación** El Lema de Bombeo es una propiedad que todo lenguaje regular debe cumplir.

**Uso** Para demostrar que lenguajes no son regulares, usando demostración por contradicción.

**Teorema 3.2.** El lenguaje

$$L = \{a^k b^k : k \geq 0\}$$

no es regular.

*Demostración.* Supongamos que  $L$  es regular, por lo que cumple el lema del bombeo.

- Sea  $N$  la constante del lema de bombeo (Queremos comprobar que el lema de bombeo no se cumple, o sea, tengo que demostrar que para ningún  $N$  sirve).
- Elegimos  $\sigma = a^N b^N$ ,  $|\sigma| = 2N \geq N$  (el lema asegura que todo  $\sigma \in L$ ,  $|\sigma| \geq N$  funciona, puedo elegir uno cómodo).
- Por el lema de bombeo,  $\sigma = \alpha\beta\gamma$ ,  $|\alpha\beta| \leq N$ ,  $\beta \neq \epsilon$ , y para todo  $k \geq 0$ ,  $\alpha\beta^k\gamma \in L$ . (Nótese que no tengo ningún control sobre  $\alpha$ ,  $\beta$ ,  $\gamma$ . Si usted “fija” alguna de estas particiones, su demostración no servirá).
- En este caso, significa que  $\alpha \in a^*$ ,  $\beta \in a^+$ . O sea,  $\alpha\gamma$  tiene menos  $a$  que  $b$ ,  $\alpha\gamma \notin L$ . (Puedo elegir  $k$  (el lema dice que todo  $k \geq 0$  funciona... basta con encontrar uno que falle). Nótese que si  $k = 1$ , el cuento no tendrá ninguna gracia.)

Si elegimos  $k = 0$ , vemos que la cantidad de  $a$ 's superará la cantidad de  $b$ 's, por lo tanto,  $\alpha\beta^0\gamma \notin L$ . Sin embargo, en un comienzo supusimos que  $L$  era regular. Entonces, por contradicción se tiene que  $L$  no es regular.  $\square$

**Teorema 3.3.**  $L = \{a^{k^2} : k \geq 0\}$  no es regular.

*Demostración.* Por contradicción. Supongamos  $L$  regular, y sea  $N$  la constante del lema de bombeo. Elegimos  $\sigma = a^{N^2}$ ,  $|\sigma| = N^2 \geq N$ . El lema asegura que existen  $\alpha, \beta, \gamma$ ;  $\alpha\beta\gamma = \sigma$ ,  $|\alpha\beta| \leq N$ ,  $\beta \neq \epsilon$ , tales que  $\alpha\beta^k\gamma \in L$  para todo  $k \geq 0$ . Considerando únicamente los largos, sabemos

$$0 < |\beta| \leq N$$

Sea  $u = |\beta|$ . Entonces:

$$0 < u \leq N$$

$$\underbrace{N^2 - u}_{|\alpha\gamma|} + \underbrace{ku}_{|\beta^k|} \quad \text{es un cuadrado para todo } k \geq 0$$

$$N^2 + (k-1)u \quad \text{es cuadrado para } k \geq 0$$

Consideremos  $k = 2$ . Entonces

$$N^2 + u$$

es un cuadrado, pero

$$N^2 < N^2 + u \leq N^2 + N \leq N^2 + 2N + 1 = (N + 1)^2$$

Por lo tanto,  $N^2 + u$  no es cuadrado. Luego, por contradicción, se tiene que  $L$  no es regular.  $\square$

### 3.1.1. Estrategia de demostración usando el Lema de Bombeo

Estrategia de lema de bombeo para demostrar lenguaje no es regular: juego con un adversario, que elige los peores casos para  $\exists$ , uno elige a gusto para  $\forall$  (Cuadro 3.1).

Sansano	Adversario
Propone $L$	
Elige $\sigma \in L,  \sigma  \geq N$	Pone $N$ , constante del lema
Elige $k = 0$ o $k \geq 2$ tal que $\alpha\beta^k\gamma \notin L$	Divide $\sigma = \alpha\beta\gamma,  \alpha\beta  \leq N, \beta \neq \epsilon$
	#\$!@5 sansano, me ganó de nuevo... ya verá la próxima

Cuadro 3.1: Estrategia para demostraciones con lema de bombeo.  
En esta ocasión, presentamos la intensa pelea entre el sansano y su vil adversario.

**Teorema 3.4.**  $L = \{a^p : p \text{ primo}\}$  no es regular.

*Demostración.* Elegimos  $\sigma = a^p$ , con  $p \leq N$  primo. Basta considerar largos:

$$|\beta| = u, \quad 1 \leq u \leq N \leq p$$

Por el lema de bombeo, es primo:

$$p + (k - 1)u$$

para todo  $k \geq 0$ .

Si elegimos  $k = p + 1$ , es primo:

$$p + (p + 1 - 1)u = p(1 + a)$$

y  $u + 1 \geq 2$ . Entonces, por contradicción se tiene que  $L$  no es regular.  $\square$

## 3.2. Propiedades de Clausura

**Teorema 3.5.** Si  $L_1, L_2$  son regulares, son regulares  $L_1 \cup L_2, L_1 \cdot L_2, L_1^*, L_1^+$ .

*Demostración.* Todas las operaciones anteriores se pueden demostrar a través de expresiones regulares.  $\square$

**Teorema 3.6.** Si  $L_1$  y  $L_2$  son regulares, entonces  $L_1 \cap L_2$  es regular.

*Demostración I.* Consideremos DFA's  $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  y  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$  y  $L_1 = \mathcal{L}(M_1)$ ,  $L_2 = \mathcal{L}(M_2)$ . La intersección entre ambos DFA's no es más que

$$M = (Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), F_1 \times F_2),$$

con  $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$  para todo  $p \in Q_1$ ,  $q \in Q_2$ ,  $a \in \Sigma$ . Entonces  $L_1 \cap L_2 = \mathcal{L}(M)$ .  $\square$

*Demostración II.* Usando el teorema 3.5 y el teorema 3.7: Si  $L_1$  y  $L_2$  son regulares, son regulares  $\bar{L}_1$  y  $\bar{L}_2$ , y:

$$\overline{(\bar{L}_1 \cup \bar{L}_2)} = L_1 \cap L_2$$

Por lo tanto, los lenguajes regulares son cerrados respecto a la intersección.  $\square$

**Teorema 3.7.** Si  $L$  es regular, entonces  $\bar{L}$  es regular.

*Demostración.* Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA, con  $L = \mathcal{L}(M)$ . Entonces el DFA  $\bar{M} = (Q, \Sigma, \delta, q_0, Q \setminus F)$   $\square$

**Observación:** La demostración del teorema 3.7 no se puede hacer con NFA, porque existirán palabras que un NFA acepte, pero su complemento no necesariamente aceptará lo que el NFA no acepta, dado que no está determinado el camino que sigue.

**Teorema 3.8.** Los lenguajes regulares son cerrados respecto a la diferencia de conjuntos (por ejemplo,  $A \setminus B$ )

*Demostración (vía propiedades).* Es sabido que:

$$L_1 \setminus L_2 = \overline{(\bar{L}_1 \cap L_2)} \cap L_1$$

Los lenguajes regulares son cerrados respecto a su complemento e intersección. Por lo tanto  $L_1 \setminus L_2$  es regular.  $\square$

*Demostración vía autómatas.* Sean  $L_1 = \mathcal{L}(M_1)$  y  $L_2 = \mathcal{L}(M_2)$  tales que

$$M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$$

$$M_2 = (Q_2, \Sigma, \delta_2, p_0, F_2)$$

Dado lo anterior, podemos construir un DFA

$$M' = (Q_1, \Sigma, \delta, q_0, F_1 \setminus F_2)$$

con

$$\hat{\delta}(q_0, \sigma) \in F_1 \setminus F_2 \quad ; \sigma \in \Sigma^*$$

Pero  $\mathcal{L}(M') = L_1 \setminus L_2$ . Por lo tanto,  $L_1 \setminus L_2$  es regular.  $\square$

**Teorema 3.9.** Los lenguajes regulares son cerrados respecto a la diferencia simétrica (por ejemplo,  $A \Delta B$ )

*Demostración vía propiedades.* Por definición:

$$L_1 \Delta L_2 = (L_1 \cup L_2) \setminus (L_1 \cap L_2)$$

Los lenguajes regulares son cerrados respecto a la unión, intersección y diferencia de conjuntos. Por lo tanto  $L_1 \Delta L_2$  es regular.  $\square$

*Demostración vía autómatas.* Sean  $L_1 = \mathcal{L}(M_1)$  y  $L_2 = \mathcal{L}(M_2)$  tales que

$$M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$$

$$M_2 = (Q_2, \Sigma, \delta_2, p_0, F_2)$$

Dado lo anterior, podemos construir un DFA

$$M' = (Q_1, \Sigma, \delta, q_0, F_1 \Delta F_2)$$

con

$$\hat{\delta}(q_0, \sigma) \in F_1 \Delta F_2 \quad ; \sigma \in \Sigma^*$$

Pero  $\mathcal{L}(M') = L_1 \Delta L_2$ . Por lo tanto,  $L_1 \Delta L_2$  es regular.  $\square$

**Definición 3.1** (Substitución). Sea  $L$  un lenguaje sobre  $\Sigma$  y sean  $L_a$  lenguajes sobre  $\Delta$ , para todo  $a \in \Sigma$ . La substitución  $\sigma$  dada por  $L_a$  se define:

$$a_1 a_2 \dots a_n \mapsto L_{a_1} L_{a_2} \dots L_{a_n}$$

Esto puede extenderse a lenguajes en forma natural:

$$\sigma(L) = \bigcup_{\alpha \in L} \sigma(\alpha)$$

**Observación:** La substitución reemplaza cada símbolo de  $\Sigma$  por un lenguaje sobre  $\Delta$ . Nótese que este lenguaje sobre  $\Delta$  puede ser regular o no regular. No hay restricciones.

**Teorema 3.10.** Los lenguajes regulares son cerrados respecto a la substitución.

*Demostración.* Sean  $L, L_a$  regulares para  $a \in \Sigma$ . Supongamos R.E.s  $R$  y  $R_a$  tales que:

$$L = \mathcal{L}(R)$$

$$L_a = \mathcal{L}(R_a)$$

para todo  $a \in \Sigma$ . Substituyendo  $R_a$  para  $a$  en  $R$ , obtenemos una R.E. para  $\sigma(L)$ .  $\square$

**Definición 3.2** (Homomorfismo). Un homomorfismo de  $\Sigma^*$  a  $\Delta^*$  es una función  $h : \Sigma^* \rightarrow \Delta^*$  tal que  $h(\alpha\beta) = h(\alpha)h(\beta)$  para todo  $\alpha, \beta \in \Sigma^*$

Observaciones de un homomorfismo:

- Es claro que  $h(\epsilon) = \epsilon$ , y por  $h(a_1 a_2 \dots a_n) = h(a_1) h(a_2) \dots h(a_n)$  y basta definir  $h$  para  $a \in \Sigma$ .
- Un homomorfismo preserva una operación. Para lo que estamos viendo preserva la operación de concatenación.
- Un homomorfismo transforma cada símbolo de  $\Sigma$  por una palabra sobre  $\Delta^*$

**Teorema 3.11.** Los lenguajes regulares son cerrados respecto al homomorfismo.

*Demostración.* Un homomorfismo es una substitución de un lenguaje de una palabra.  $\square$

**Teorema 3.12.** Los lenguajes regulares son cerrados respecto de homomorfismo inverso.

*Demostración.* Sea  $L \subseteq \Delta^*$  un lenguaje regular,  $h : \Sigma^* \rightarrow \Delta^*$  un homomorfismo. Sea  $M = (Q, \Delta, \delta, q_0, F)$  un DFA tal que  $L = \mathcal{L}(M)$ . El DFA  $M' = (Q, \Sigma, \delta', q_0, F)$  con  $\delta'(q, a) = \delta(q, h(a))$  para todo  $q \in Q, a \in \Sigma$  acepta  $h^{-1}(L)$ .  $\square$

### 3.3. Un lenguaje que cumple el Lema de Bombeo sin ser regular

**Teorema 3.13.** El lenguaje  $L = \{ab^n c^n : n \geq 0\} \cup \{a^i b^j c^k : i \neq 1 \wedge j \geq 0 \wedge k \geq 0\}$  cumple el lema de bombeo, pero no es regular.

*Demostración.* La conclusión del Lema del Bombeo asegura que toda palabra  $\sigma \in L$  con  $|\sigma| \geq N$  (suponiendo que  $L$  es un lenguaje regular) se puede particionar en  $\sigma = \alpha\beta\gamma$  tal que  $|\beta| > 0$ ,  $|\alpha\beta| \leq N$  y  $\alpha\beta^k\gamma \in L$ .

i.  $\sigma$  comienza con  $a$ :

Si bombeamos  $a$ , queda  $a^k b^n c^n \in L$ .

Si  $\sigma = a^r b^s c^k$  con  $r \geq 2$ , al bombear la  $a$  queda  $a^{r+k-1} b^s c^k \in L$ , porque al ser  $k \geq 0$ ,  $r \geq 2$  queda  $r+k-1 \geq 1$ .

ii.  $\sigma$  comienza con  $b$  o  $c$ : No hay problema en bombear, queda algo en  $b^* c^* \in L$

$\leadsto$  cumple las conclusiones del lema de bombeo.

Pero  $L$  no es regular. Consideremos el homomorfismo:

- $h(a) = \epsilon$
- $h(b) = a$
- $h(c) = b$

Consideremos:

$$h(L \cap \mathcal{L}(ab^*c^*)) = \{a^n b^n : n \geq 0\}$$

que sabemos que no es regular. Si  $L$  fuera regular, esto sería regular. Entonces, por contradicción se tiene que  $L$  no es regular.  $\square$

### 3.4. Ejercicios

**Ejercicio 3.1.** Sea  $L$  un lenguaje que está compuesto por todos los factoriales de un número binario. Demuestre que no es regular.

**Hint:** considere los largos.



## Capítulo 4

# Pushdown Automata

**Idea:** NFA + Stack como memoria auxiliar.

### 4.1. Definición de un Autómata de Stack

---

**Definición 4.1.** Un PDA (o autómata de pila/stack)  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  consta de:

- $Q$ : conjunto finito de estados.
- $\Sigma$ : alfabeto de entrada.
- $\Gamma$ : Alfabeto de stack.
- $\delta$ : función de transición:

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \underbrace{2^{Q \times \Gamma^*}}_{\text{subconjuntos finitos}}$$

- $q_0$ : estado inicial,  $q_0 \in Q$ .
- $Z_0$ : stack inicial,  $Z_0 \in \Gamma$
- $F$ : estados finales,  $F \subseteq Q$ .

**Idea:** Si  $M$  está en el estado  $q$ , y en el tope del stack tiene  $A$ , al leer  $x$  ( $x \in \Sigma$ , o  $x = \epsilon$ ) de la entrada, si  $(p, \gamma) \in \delta(q, x, A)$  entonces  $M$  puede pasar al estado  $p$ , consume  $x$  de la entrada, y reemplaza  $A$  por  $\gamma$  en el stack.

#### 4.1.1. Notación Gráfica de un Autómata de Stack

De la misma forma que vimos con los DFA y NFA, podemos representar los PDA a través de un grafo dirigido, donde:

- a) Los vértices son los estados del PDA.
- b) El arco etiquetado con “Inicio” corresponde al *estado inicial* del PDA, y todos aquellos que tienen un doble circulo son considerados *estados finales* (Figura 4.1).
- c) Los arcos corresponden a las transiciones del PDA, bajo la notación

$$x, A/B \tag{4.1.1}$$



Figura 4.1:  $q_0$  representa un estado inicial,  $q_n$  es un estado cualquiera y  $q_f$  es un estado final.

donde  $x$  es un símbolo,  $A$  y  $B$  son símbolos del stack. La notación (4.1.1) se lee como: “si leo el símbolo  $x$  y mi tope del stack es  $A$ , realizo una transición a otro estado (que puede ser el mismo) reemplazando el tope del stack  $A$  por  $B$ ”.

**Ejemplo 4.1.** La Figura 4.2 muestra un ejemplo de un PDA. Repasemos la notación de las transiciones

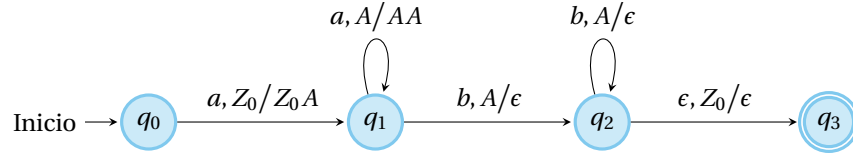


Figura 4.2: Ejemplo de un PDA en forma de digrafo.

de un PDA descrita en (4.1.1) y además, consideremos únicamente los estados  $q_0$  y  $q_1$  para simplificar (Figura 4.3). Si estamos en el estado  $q_0$ , podremos acceder a  $q_1$  únicamente si leemos el símbolo  $a$  y

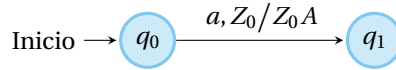


Figura 4.3: Otro PDA

al mismo tiempo, si  $Z_0$  es el tope del stack. En ese caso, pasamos a  $q_1$  consumiendo  $a$  y reemplazando el tope del stack (que en ese momento es  $Z_0$ ) por  $Z_0A$ .

#### 4.1.2. Descripción instantánea y lenguajes de un PDA

**Definición 4.2.** Una *descripción instantánea* del PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  es:

$$(\alpha q \beta, \gamma) \quad (4.1.2)$$

con  $\alpha, \beta \in \Sigma^*$  (entrada leída y por leer)  $q \in Q$  (estado actual),  $\gamma \in \Gamma^*$  (stack actual).

**Definición 4.3.**  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  un PDA. La *relación de transición de  $M$* ,  $\vdash_M$  entre ID's de  $M$  se define por:

- Si  $(p, \sigma) \in \delta(q, \epsilon, A)$  entonces:

$$(\alpha q \beta, \gamma A) \vdash_M (\alpha p \beta, \gamma \sigma) \quad (4.1.3)$$

- Si  $(p, \sigma) \in \delta(q, a, A)$ :

$$(\alpha q a \beta, \gamma A) \vdash_M (\alpha p \beta, \gamma \sigma) \quad (4.1.4)$$

Se suele escribir  $\vdash$  si  $M$  se subentiende.

**Definición 4.4.** Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  un PDA. El *lenguaje aceptado por  $M$  por estado final* es:

$$\mathcal{L}(M) = \{\sigma : (q_0 \sigma, Z_0) \vdash_M^* (\sigma q_f, \gamma) \wedge \gamma \in N \wedge q_f \in F\} \quad (4.1.5)$$

**Definición 4.5.** Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  un PDA. El lenguaje aceptado por stack vacío por  $M$  es:

$$\mathcal{N}(M) = \{\sigma : (q_0\sigma, Z_0) \vdash_M^* (\sigma q, \epsilon)\} \quad (4.1.6)$$

En general,  $\mathcal{L}(M) \neq \mathcal{N}(M)$  (los lenguajes aceptados por estado final y por stack vacío son diferentes). Al diseñar PDA para aceptar por stack vacío, por convención se da  $F = \emptyset$ .

**Definición 4.6.** El PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  se dice determinista (DPDA) si:

- I. Para todo  $q \in Q$ ,  $x \in (\Sigma \cup \{\epsilon\})$ ,  $A \in \Gamma$  es  $|\delta(q, x, A)| \leq 1$ , es decir, el PDA tiene a lo más 1 movida posible.
- II. Si para todo  $a \in \Sigma$ ,  $\delta(q, a, A) \neq \emptyset$ , entonces  $\delta(q, \epsilon, A) = \emptyset$ . Con esto, forzamos al PDA para que tenga sólo una movida.

**Teorema 4.1.** Hay lenguajes aceptados por PDA que **no** son aceptados por DPDA.

*Demostración.* Consideremos el lenguaje de (4.1.7).

$$L = \{\sigma\sigma^R : \sigma \in \{a, b\}^*\} \quad (4.1.7)$$

La Figura 4.4 representa un PDA que reconoce  $L$  por stack vacío. Este PDA no es determinista:

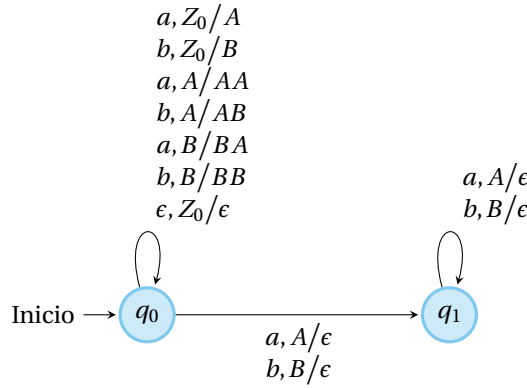


Figura 4.4: En el loop del estado  $q_0$  llenamos el stack del PDA. Luego, al pasar de  $q_0 \rightarrow q_1$  incluyendo el loop, vaciamos el stack.

$$\delta(q_0, a, A) = \{(q_0, AA), (q_1, \epsilon)\}$$

$$\delta(q_0, b, B) = \{(q_0, BB), (q_1, \epsilon)\}$$

Informalmente,  $\mathcal{N}(M) = L$ .

Pero ningún DPDA puede aceptar  $L$  por stack vacío. Como  $\epsilon \in L$ , después de la primera movida, un DPDA queda con stack vacío, y no consume el resto si  $\sigma \in L$  con  $\sigma \neq \epsilon$ .  $\square$

**Observación:** La demostración formal del teorema 4.1 se puede hacer por inducción. El caso base es la palabra  $\sigma$ .

**Teorema 4.2.** Si  $L = \mathcal{L}(M)$ , para  $M$  un PDA, podemos construir un PDA  $M'$  tal que  $L = \mathcal{N}(M')$ .

*Demostración.* La idea consiste en consumir el stack al llegar a un estado final. Evitar aceptación “accidental” por  $Z'_0$  bajo el stack de  $M$ .

Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ . Definimos  $M' = (Q \cup \{q'_0, q'_f\}, \Sigma, \Gamma, \delta', q'_0, Z'_0, \emptyset)$ , donde:

$$\delta'(q'_0, \epsilon, Z'_0) = \{(q_0, Z'_0 Z_0)\} \quad (4.1.8)$$

Todos las movidas de  $M$ :

$$\delta'(q, x, A) = \delta(q, x, A) \quad (4.1.9)$$

siempre que  $q \neq F$

$$\delta'(q_f, \epsilon, A) = \delta(q_f, \epsilon, A) \cup \{(q'_f, \epsilon)\} \quad (4.1.10)$$

para  $q_f \in F$ . Además, para todo lo anterior considere  $x \in \Sigma \cup \{\epsilon\}$ ,  $q \in Q \setminus F$ ,  $q_f \in F$ ,  $A \in \Gamma$ ,  $a \in \Sigma$ .

$$\delta'(q'_f, \epsilon, A) = \{(q'_f, \epsilon)\} \text{ para } A \in \Gamma \cup \{Z'_0\} \quad \square$$

**Teorema 4.3.** Si  $L = \mathcal{N}(M)$ , podemos construir  $M'$  con  $L = \mathcal{L}(M')$

*Demostración.* Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ . Definimos

$$M' = (Q \cup \{q'_0, q'_f\}, \Sigma, \Gamma \cup \{Z'_0\}, \delta', q'_0, Z'_0, \{q'_f\})$$

con:

$$\begin{aligned} \delta'(q'_0, Z'_0) &= \{(q_0, Z'_0 Z_0)\} \\ \delta'(q, x, A) &= \delta(q, x, A) \quad \text{para } q \in Q, x \in \Sigma \cup \{\epsilon\}, A \in \Gamma \\ \delta'(q, \epsilon, Z'_0) &= \{(q'_f, \epsilon)\} \quad \text{para } q \in Q \end{aligned}$$

$\square$

## Capítulo 5

# Gramáticas y Lenguajes de Contexto Libre

### 5.1. Gramáticas

---

**Definición 5.1.** Una gramática  $G = (N, \Sigma, P, S)$  consta de:

- $N$ : El alfabeto de no-terminales. (es una  $\nu$  (nu) mayúscula, no una  $n$ )
- $\Sigma$ : El alfabeto de terminales.
- $P$ : conjunto de producciones de la forma  $\alpha \rightarrow \beta$  con  $\alpha \in (N \cup \Sigma)^*$ , conteniendo al menos un símbolo de  $N$ ,  $\beta \in (N \cup \Sigma)^*$
- $S$ : símbolo de partida,  $S \in N$ .

Notar que  $N \cap \Sigma = \emptyset$  y  $N \cup \Sigma = V$  (vocabulario).

**Definición 5.2.** Sea  $G = (N, \Sigma, P, S)$  una gramática. Definimos la relación de derivación de  $G$  sobre  $(N \cup \Sigma)^*$  a través de:

$$u\alpha v \Rightarrow_G u\beta v$$

para  $u, v \in (\Sigma \cup N)^*$  si  $\alpha \rightarrow \beta \in P$

**Definición 5.3.** Sea  $G = (N, \Sigma, P, S)$  una gramática. Se define el lenguaje generado por  $G$  como:

$$\mathcal{L}(G) = \{\sigma : S \Rightarrow_G^* \sigma \wedge \sigma \in \Sigma^*\} \quad (5.1.1)$$

#### 5.1.1. Jerarquía de Chomsky

Sea  $G = (N, \Sigma, P, S)$  una gramática. Se dice que es:

- **Tipo 0 (irrestringida)** – Como definido.
- **Tipo 1 (sensibles al contexto)** – Toda las producciones  $\alpha \rightarrow \beta$  cumplen:

$$|\alpha| \leq |\beta| \quad ; \text{ las palabras nunca se acortan} \quad (5.1.2)$$

Nótese que  $\alpha \neq \epsilon$  por definición del conjunto de producciones. (Una manera alternativa de describirlas es decir que las producciones tienen la forma  $\alpha A \beta \rightarrow \alpha \gamma \beta$ , con  $\alpha, \beta \in (N \cup \Sigma)^*$ ,  $A \in N$ ,  $\gamma \in (N \cup \Sigma)^+$ )

Acá sólo podemos reemplazar la producción  $A$  por  $\gamma$  sólo si lo rodean  $\alpha$  y  $\beta$ ... por eso es sensible al contexto...

- **Tipo 2 (contexto libre)** – Todas las producciones tienen al forma  $A \rightarrow \alpha$ , con  $A \in N$ ,  $\alpha \in (N \cup \Sigma)^+$
- **Tipo 3 (regular)** – Todas las producciones sólo pueden ser de la forma:

$$\begin{aligned} A &\rightarrow \alpha B, & \text{con } A, B \in N, \alpha \in \Sigma^* \\ A &\rightarrow \alpha, & \text{con } A \in N, \alpha \in \Sigma^+ \end{aligned}$$

**Observación:** Las gramáticas de tipo 1 son un caso particular de tipo 0, las de tipo 2 particulares de tipo 1 y así sucesivamente.

**Teorema 5.1.** Las gramáticas de tipo 3 generan los lenguajes regulares (gramática tipo 3  $\Leftrightarrow$  lenguaje regular).

*Demostración.* Comenzamos demostrando

$$\text{lenguaje regular} \Rightarrow \text{gramática tipo 3 (sin } \epsilon) \quad (5.1.3)$$

$L$  regular  $\Rightarrow L = \mathcal{L}(M)$ , con  $M$  un DFA tal que el estado inicial no es final. Sea  $M = (Q, \Sigma, \delta, q_0, F)$  el DFA. Sin pérdida de generalidad  $Q \cap \Sigma = \emptyset$  (la idea es usar los estados como no terminales).

Definimos la gramática  $G = (Q, \Sigma, P, q_0)$ . Las producciones son:

- Si  $\delta(q, a) = p$ , entonces  $q \rightarrow ap$
- Si  $\delta(q, a) = p$ , y  $p$  es final,  $q \rightarrow a$ .

$$\therefore \mathcal{L}(G) = \mathcal{L}(M).$$

Ya demostrado (5.1.3), nos falta demostrar que

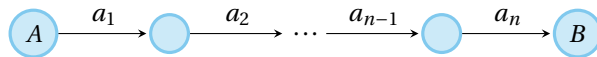
$$\text{gramática tipo 3} \Rightarrow \text{lenguaje regular} \quad (5.1.4)$$

Sea  $G = (N, \Sigma, P, S)$  una gramática de tipo 3. Construimos un NFA que acepte  $\mathcal{L}(G)$  vía:

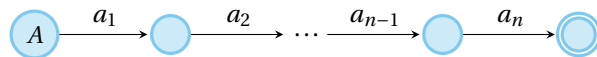
$$M = (N \cup \{q_1, \dots\}, \Sigma, S, F)$$

con:

- Si  $A \rightarrow a_1 a_2 \dots a_n B$ :



- Si  $A \rightarrow a_1 a_2 \dots a_n$ :



Con esto, queda demostrado (5.1.4).

Finalmente, se tiene que las gramáticas de tipo 3 generan lenguajes regulares y viceversa. □

Dado que las gramáticas de tipo 3 sólo generan lenguajes regulares, podemos modificar nuestro triángulo NFA - DFA - RE agregándole un “cototo”:

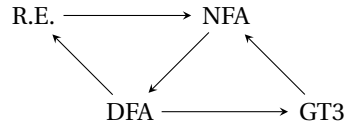


Figura 5.1: GT3 son las gramáticas de tipo 3 son aquellas que generan R.E.

## 5.2. Gramáticas de Contexto Libre

Sea  $G = (N, \Sigma, P, S)$  una gramática de contexto libre (CFG). Algunas convenciones:

- Mayúsculas son no-terminales ( $A, B, \dots, S, \dots$ )
- Otros símbolos (minúsculas, ..., ) son terminales.
- Se anotan sólo las producciones, el símbolo no terminal de la primera es el símbolo de partida.

Ejemplo (ambas producciones forman parte de la misma gramática):

$$S \rightarrow ab$$

$$S \rightarrow aSb$$

Vemos que:

$$S \Rightarrow ab$$

$$S \Rightarrow aSb$$

$$\Rightarrow a^2 S b^2$$

$$\vdots$$

$$\Rightarrow a^{n-1} S b^{n-1}, \quad n \in \mathbb{N}$$

$$\Rightarrow a^n b^n, \quad n \in \mathbb{N}$$

$\therefore$  Hay gramáticas de contexto libre que generan lenguajes que NO son regulares.

Por lo general, cuando hablemos de gramáticas, sólo nombraremos el conjunto de producciones que esta trae para simplificar.

### 5.2.1. Ejemplo informal de una Gramática de Contexto Libre

Un ejemplo de gramática puede ser:

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow (E)$$

$$F \rightarrow a$$

Como puede ver, el CFG anterior representa la sintaxis que se debe seguir al resolver una expresión algebraica. Es muy similar al BNF.

Nótese que existen otras CFG equivalentes a las que se mencionan en el párrafo anterior, por ejemplo:

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$E \rightarrow a$$

También puede ser:

$$E \rightarrow E + T$$

$$E \rightarrow E * T$$

$$E \rightarrow T$$

$$T \rightarrow (E)$$

$$T \rightarrow a$$

### 5.3. Árboles de Derivación

**Definición 5.4.** /\* Colocar la definición de derivación \*/

**Definición 5.5.** Sea  $G = (N, \Sigma, P, S)$  un CFG. El árbol de derivación de  $\sigma \in \mathcal{L}(G)$ . Se construye poniendo el símbolo de partida como raíz, y bajo cada no-terminal los símbolos que deriva (en orden).

**Ejemplo 5.1.** Consideremos la siguiente gramática:

$$E \rightarrow E + T \quad (1)$$

$$E \rightarrow T \quad (2)$$

$$T \rightarrow T * F \quad (3)$$

$$T \rightarrow F \quad (4)$$

$$F \rightarrow (E) \quad (5)$$

$$F \rightarrow a \quad (6)$$

Queremos ver los pasos por los que tenemos que pasar para llegar a  $a * (a + a)$  (es sólo un ejemplo). Para ello considere lo siguiente:

- Colocamos el no-terminal inicial que nos ayude a armar el cuento.
- Anotaremos la producción que usaremos en el lado derecho de la ecuación para no perdernos.
- Subrayamos el no-terminal que vamos a reemplazar.
- Reemplazamos.



Entonces:

$$\begin{aligned}
 E &\Rightarrow \underline{T} & (2) \\
 &\Rightarrow T * \underline{F} & (3) \\
 &\Rightarrow T * (\underline{E}) & (5) \\
 &\Rightarrow T * (\underline{E} + T) & (1) \\
 &\Rightarrow T * (T + \underline{T}) & (2) \\
 &\Rightarrow T * (T + \underline{F}) & (4) \\
 &\Rightarrow \underline{T} * (T + a) & (6) \\
 &\Rightarrow F * (\underline{T} + a) & (4) \\
 &\Rightarrow F * (\underline{F} + a) & (4) \\
 &\Rightarrow \underline{F} * (a + a) & (6) \\
 &\Rightarrow a * (a + a) & (6)
 \end{aligned}$$

Luego, el árbol de derivación del caso sería:

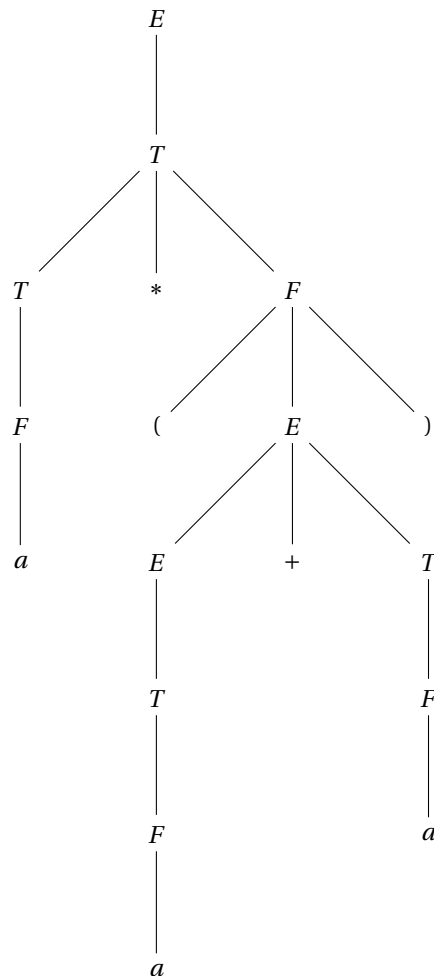


Figura 5.2: Árbol de Derivación resultante de las *relaciones de derivaciones* ( $\Rightarrow$ ) aplicadas en el ejemplo anterior.

**Definición 5.6** (Derivación de extrema izquierda). En cada iteración aplicamos la producción al no-terminal del extremo izquierdo. Para la gramática del ejemplo 5.1, la derivación de extrema izquierda

es:

$$\begin{aligned}
 E &\Rightarrow \underline{T} \\
 &\Rightarrow \underline{T} * F \\
 &\Rightarrow \underline{F} * F \\
 &\Rightarrow a * \underline{F} \\
 &\Rightarrow a * (\underline{E}) \\
 &\Rightarrow a * (\underline{E} + T) \\
 &\Rightarrow a * (\underline{T} + T) \\
 &\Rightarrow a * (\underline{F} + T) \\
 &\Rightarrow a * (a + \underline{T}) \\
 &\Rightarrow a * (a + \underline{F}) \\
 &\Rightarrow a * (a + a)
 \end{aligned}$$

**Definición 5.7** (Derivación de extrema derecha). En cada iteración, aplicamos la producción al no-terminal del extremo derecho. Para la gramática del ejemplo 5.1, la derivación de extrema derecha es:

$$\begin{aligned}
 E &\Rightarrow \underline{T} \\
 &\Rightarrow T * \underline{F} \\
 &\Rightarrow T * (\underline{E}) \\
 &\Rightarrow T * (\underline{E} + T) \\
 &\Rightarrow T * (\underline{E} + \underline{F}) \\
 &\Rightarrow T * (\underline{E} + a) \\
 &\Rightarrow T * (\underline{T} + a) \\
 &\Rightarrow T * (\underline{F} + a) \\
 &\Rightarrow \underline{T} * (a + a) \\
 &\Rightarrow \underline{F} * (a + a) \\
 &\Rightarrow a * (a + a)
 \end{aligned}$$

Algunas observaciones:

- Si podemos derivar una palabra, entonces podemos construir un árbol de derivación.
- Para encontrar la palabra del árbol de derivación debemos recorrerlo en **post-orden** (más detalles en sus apuntes de estructura de datos). Esta serán los no-terminales en el orden que los vayamos encontrando.
- Sólo en las gramáticas de contexto libre podemos aplicar las producciones en distinto orden (extrema izquierda o derecha por ejemplo), ya que son libres al contexto (valga la redundancia). Como es de esperar, en un lenguaje sensible al contexto esto no ocurre.

## 5.4. Ambigüedad en Gramáticas y Lenguajes

Árbol de derivación  $\equiv$  derivación de extrema izquierda  $\equiv$  derivación de extrema derecha (son equivalentes).

$\rightsquigarrow$  Colección de derivaciones que “hacen lo mismo”

**Definición 5.8** (Gramática Ambigua). Sea  $G = (N, \Sigma, P, S)$  una CFG. Si hay  $\sigma \in \mathcal{L}(G)$  con dos árboles de derivación distintos, se dice que  $G$  es ambigua.

**Ejemplo 5.2.** Consideremos la gramática  $G = (N, \Sigma, P, E)$  con el siguiente conjunto de producciones:

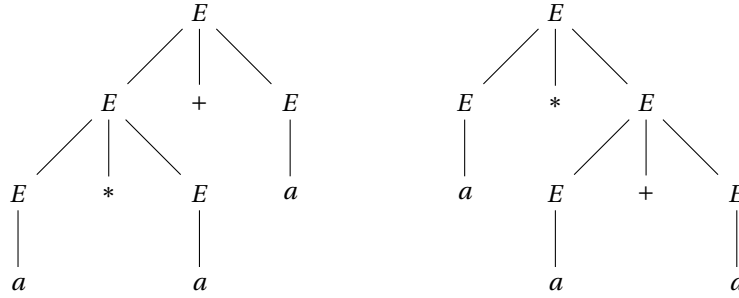
$$E \rightarrow E + E \quad (1)$$

$$E \rightarrow E * E \quad (2)$$

$$E \rightarrow (E) \quad (3)$$

$$E \rightarrow a \quad (4)$$

Supongamos además, que queremos formar la palabra  $a * a + a$  con la gramática  $G$ . Si comenzamos a derivar usando (1), obtendremos el árbol de la Figura 5.3a; sin embargo, si empezamos a derivar usando (2), el árbol de derivación resultante sería el de la Figura 5.3b.



(a) En esta ocasión, comenzamos aplicando la producción (1) (b) Para este caso, comenzamos aplicando la producción (2)

Figura 5.3: Gramática ambigua del ejemplo 5.2.

La Figura 5.3, deja en manifiesto que para aplicaciones prácticas se debe tener cuidado con los árboles de derivación. Si la gramática ambigua, entonces no sirve (por ejemplo, una palabra con varios significados posibles... considere la gramática del ejemplo 5.2)

**Ejercicio 5.1.** Demuestre que existen lenguajes regulares generados por gramáticas de tipo 2 y que no son de tipo 3.

*Demostración.* Sea  $G = (N, \Sigma, P, S)$  con las siguientes producciones:

$$S \rightarrow aSb$$

$$S \rightarrow ab$$

$$S \rightarrow aa$$

$$S \rightarrow ba$$

$$S \rightarrow bb$$

La gramática anterior es el lenguaje regular  $\mathcal{L}(a^*(a|b)(b|a)b^*)$ . □

**Ejercicio 5.2.** Demuestre que el lenguaje generado por la gramática

$$E \rightarrow E + E \quad (1)$$

$$E \rightarrow E * E \quad (2)$$

$$E \rightarrow (E) \quad (3)$$

$$E \rightarrow a \quad (4)$$

no es regular.

Algunos comentarios al margen antes de terminar:

- Determinar si dos lenguajes de contexto libre generan el mismo lenguaje es imposible
- La libertad de cómo aplicar las producciones en una CFG, se representa a través del árbol de derivación

## 5.5. Equivalencia entre PDA y CFG

Con las herramientas que disponemos actualmente, nos encontramos listos para demostrar que los lenguajes definidos por un PDA son lenguajes de contexto libre. El plan de ataque se muestra en la Figura 5.4. Nuestra meta es demostrar que los siguientes lenguajes:

1. Los lenguajes de contexto libre, lenguajes (valga la redundancia) definidos por CFG.
2. Los lenguajes que son aceptados por un PDA que acepte por estado final.
3. Los lenguajes que son aceptados por un PDA que acepte por stack vacío.

son lo mismo. Previamente, ya demostramos que los lenguajes aceptados por un PDA que acepte por stack vacío y por estado final son el mismo (véase los teoremas 4.2 y 4.3). Por lo tanto, sólo tenemos que demostrar que los lenguajes de contexto libre y aquellos que son generados por un PDA que acepte por estado final son el mismo.

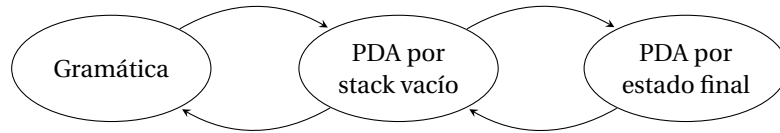


Figura 5.4: Organización de construcciones que muestran la equivalencia de las tres formas de definir CFL [1].

### 5.5.1. Transformar una CFG a un PDA

**Teorema 5.2** (CFG  $\rightarrow$  PDA). Si  $L = \mathcal{L}(G)$ , con  $G = (N, \Sigma, P, S)$  una CFG, podemos construir un PDA  $M$  tal que  $L = \mathcal{L}(M)$ .

*Demostración.* La idea es que el PDA tiene en el tope del stack la palabra después de los terminales iniciales en una forma sentencial.

Sea  $M = (\{q_0\}, \Sigma, \Sigma \cup N, \delta, q_0, S, \emptyset)$ . Acá  $\delta$  está dado por:

- $(q_0, \alpha) \in \delta(q_0, \epsilon, A)$  si  $A \rightarrow \alpha$  es producción.
- $\delta(q_0, a, a) = \{(q_0, \epsilon)\}$  para todo  $a \in \Sigma$ .

Consideremos como ejemplo, nuestra gramática regalona dada por:

$$E \rightarrow E + T \quad (1)$$

$$E \rightarrow T \quad (2)$$

$$T \rightarrow T * F \quad (3)$$

$$T \rightarrow F \quad (4)$$

$$F \rightarrow (E) \quad (5)$$

$$F \rightarrow a \quad (6)$$

Sabemos  $E \Rightarrow^* a * (a + a)$ . Podemos describirlo a través del Cuadro 5.1. Es importante destacar, que el stack del cuadro 5.1 crece hacia la izquierda ( $\leftarrow$ ).

Este PDA traza una derivación de extrema izquierda de  $\sigma$ . □

Entrada	Stack	Operación
$a * (a + a)$	$E$	$E \rightarrow T$
$a * (a + a)$	$T$	$T \rightarrow T * F$
$a * (a + a)$	$T * F$	$T \rightarrow F$
$a * (a + a)$	$F * F$	$F \rightarrow a$
$a * (a + a)$	$a * F$	shift $a$
$* (a + a)$	$* F$	shift $*$
$(a + a)$	$F$	$F \rightarrow (E)$
$(a + a)$	$F$	$F \rightarrow (E)$
$(a + a)$	$(E)$	shift $($
$a + a)$	$E)$	$E \rightarrow E + T$
$a + a)$	$E + T)$	$E \rightarrow T$
$a + a)$	$T + T)$	$T \rightarrow F$
$a + a)$	$T + T)$	$F \rightarrow a$
$a + a)$	$a + T)$	shift $a$
$+ a)$	$+ T)$	shift $+$
$a)$	$T)$	$T \rightarrow F$
$a)$	$F)$	$F \rightarrow a$
$a)$	$a)$	shift $a$
$)$	$)$	shift $)$
$\epsilon$	$\epsilon$	

Cuadro 5.1: Aplicamos producciones en el stack hasta que el tope del stack sea un no-terminal. En ese momento, aplicamos una operación shift la cual hace un pop tanto de la Entrada de no-terminales como del Stack

### 5.5.2. Transformar un PDA a una CFG

**Teorema 5.3** (PDA  $\rightarrow$  CFG). Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$  un PDA. Entonces, hay una gramática de contexto libre  $G = (N, \Sigma, P, S)$  tal que  $\mathcal{L}(G) = \mathcal{N}(M)$ .

*Demostración (informal).* La idea general de la demostración consiste en emular una gramática de contexto libre a través de un PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$  que acepte por stack vacío. Para ello, definiremos la gramática  $G = (N, \Sigma, P, S)$  tal que:

1.  $S$  corresponde al símbolo de partida.
2. Los no-terminales son  $[p, A, q]$ , con  $p, q \in Q$ , y  $A \in \Gamma$ .

Las producciones de  $G$  son las siguientes:

- a) Para todos los estados  $p \in Q$ ,  $G$  tiene producciones

$$S \rightarrow [q_0, Z_0, p]$$

Lo que se busca es eliminar  $Z_0$  del stack. Para hacerlo, es necesario pasar por una montonera de estados: comenzamos en  $q_0$  y terminamos en  $p$ . En otras palabras,  $[q_0, Z_0, p]$  es todo string  $\sigma \in \Sigma^*$  tal que  $M$  saque a  $Z_0$  del stack, es decir

$$(q_0 \sigma, Z_0) \vdash_M^* (\sigma p, \epsilon)$$

- b) Si  $(p, B_1 B_2 \dots B_n) \in \delta(q, A, x)$ , con  $x \in \Sigma \cup \{\epsilon\}$ . Entonces, para toda una lista de estados  $q_1, q_2, \dots, q_n$ ,  $G$  tiene la producción:

$$[q, A, q_n] \rightarrow x [p, B_1, q_1] [q_1, B_2, q_2] \cdots [q_{n-1}, B_n, q_n] \quad (5.5.1)$$

La producción (5.5.1) dice que la única forma de sacar  $A$  del stack e ir desde el estado  $q$  a  $q_n$  es leer  $x$ . De inmediato, usar alguna entrada para sacar  $B_1$  del stack mientras vamos del estado  $p$  al estado  $q_1$ . En seguida, remover  $B_2$  del stack a medida que vamos de  $q_1$  a  $q_2$ , y así sucesivamente hasta sacar  $B_n$  del stack mientras vamos del estado  $q_{n-1}$  a  $q_n$ .

Nótese que (5.5.1) representa sólo una movida posible del PDA... considere las demás.

- c) Si  $(p, \epsilon) \in \delta(q, A, x)$ , con  $x \in \Sigma \cup \{\epsilon\}$ , entonces:

$$[q, A, x] \rightarrow x$$

Importante destacar que si  $x = \epsilon$ , esto no es CFG. En breve corregiremos esto.

□

## Capítulo 6

# Propiedades de los Lenguajes de Contexto Libre

Completaremos nuestro estudio sobre los lenguajes de contexto libre aprendiendo alguna de sus propiedades. Nuestra primera tarea es simplificar las gramáticas de contexto libre; estas simplificaciones harán mucho más fácil demostrar características de los CFL. Luego, re-introduciremos a nuestros viejos amigos:

- Lema de Bombeo
- Propiedades de Clausura

ambos, orientados a lenguajes de contexto libre.

### 6.1. Forma normal para Gramáticas de Contexto Libre

---

La meta de esta sección es mostrar que todos los lenguajes de contexto libre (sin  $\epsilon$ ) son generados por CFG en el cual todas sus producciones son de la forma  $A \rightarrow BC$  o  $A \rightarrow a$ , donde  $A, B$  y  $C$  son no-terminales, y  $a$  es un símbolo terminal. Esta forma es llamada *Forma Normal de Chomsky* (CNF). Antes de entrar en detalles con CNF, debemos hacer un par de simplificaciones preliminares, las que serán útiles de varias maneras:

1. Debemos eliminar los *símbolos inútiles*. Estos pueden ser no-terminales o símbolos terminales, y se les llama así porque no aparecen en ninguna derivación de una palabra desde el símbolo de partida.
2. Debemos eliminar toda producción que contenga  $\epsilon$ . Estas producciones son todas aquellas que son de la forma  $A \rightarrow \epsilon$ , donde  $A$  es un no-terminal.
3. Debemos eliminar las *producciones unitarias*. Son todas aquellas que son de la forma  $A \rightarrow B$ , donde  $A$  y  $B$  son no-terminales.

#### 6.1.1. Eliminando símbolos inútiles

**Definición 6.1.** Se dice que  $X$  es *generante* en la CFG  $G = (N, \Sigma, P, S)$ , con  $X \in N \cup \Sigma$  si  $X \Rightarrow^* \alpha \in \Sigma^*$

**Definición 6.2.** Se dice que  $X$  es *alcanzable* en CFG  $G$  si  $S \Rightarrow^* \alpha X \beta$

Consideremos la CFG:

$$\begin{aligned} S &\rightarrow AB \mid a \\ A &\rightarrow b \end{aligned}$$

Todos salvo  $B$  son generantes, y todos son alcanzables.

**Observación:** Los terminales por derecho propio son generantes.

El orden en que eliminamos no generantes e inalcanzables importa:

- Primero eliminar no-generantes. Resultado:

$$\begin{aligned} S &\rightarrow a \\ A &\rightarrow b \end{aligned}$$

$\rightsquigarrow A, b$  no alcanzables. Eliminamos los inalcanzables:

$$S \rightarrow a$$

- Primero eliminar inalcanzables (no hay), luego no generantes terminamos con:

$$\begin{aligned} S &\rightarrow a \\ A &\rightarrow b \end{aligned}$$

Terminamos con basura y hasta aquí llegamos. Por ende el orden importa

Identificar símbolos generantes:

- $\Sigma \subseteq$  generantes.
- Si hay un (no-terminal)  $X$  tal que

$$X \rightarrow A_1 A_2 \cdots A_n$$

y todos los  $A_i$  con  $1 \leq i \leq n$  son generantes,  $X$  es generante.

Iterar este... Después, la gramática resultante es  $G' = (N', \Sigma, P', S)$  con:

- $N' = N \cap$  generantes
- $P'$ : producciones que no mencionan no generantes.

Caso especial: Generantes =  $\emptyset$ :

$$\rightsquigarrow G' = (\{S\}, \Sigma, \emptyset, S), \mathcal{L}(G) = \emptyset$$

Identificar alcanzables:

- $S$  es alcanzable ( $S \Rightarrow^* S$ )
- Si  $X$  es alcanzable, y

$$X \rightarrow A_1 A_2 \cdots A_n,$$

entonces  $A_1, \dots, A_n$  son alcanzables.

Iterar lo anterior, y el resultado nos dará la gramática.

$$G'' = (N'', \Sigma'', P'', S)$$

Donde:



- $N'' = N' \cap \text{alcanzables}$
- $\Sigma'' = \Sigma \cap \text{alcanzables}$
- $P''$ : producciones de  $P'$  que sólo incluyen alcanzables.

Caso especial:  $G = (\{S\}, \emptyset, \emptyset, S)$  genera  $\emptyset$ .

Algunas cosas útiles para demostrar:

- Podemos suponer que la gramática no tiene producciones unitarias.
- Podemos suponer que la gramática viene dada en forma normal de Chomsky.
- Podemos suponer que la gramática no contiene símbolos inútiles.

### 6.1.2. Eliminando producciones $\epsilon$

Pueden permitirse producciones  $A \rightarrow \epsilon$  en CFG (y en gramáticas regulares).

*Demostración.* La idea de la demostración consiste en identificar no-terminales  $A \in N$  tales que  $A \Rightarrow^* \epsilon$ ; “saltarse” esos pasos agregando versiones adicionales de los lados derechos donde aparece  $A$  sin aparecer esa  $A$  ([considere que el lenguaje original es de una gramática de tipo 0](#)).

Si  $S \Rightarrow^* \epsilon$ , muestra gramática modificada genera  $L \setminus \{\epsilon\}$  (no digamos que es un cambio tan sustancial...)

- Identificar no-terminales  $A \Rightarrow^* \epsilon$  (“nullable symbols”)

“Inducción”: para construir el conjunto  $\mathcal{A}$  (es el conjunto de nullable symbols).

**Base:** Si  $A \rightarrow \epsilon$  es una producción,  $A \in \mathcal{A}$

**Inducción:** Si  $A \rightarrow B_1 B_2 \dots B_n$  es una producción, y todos los  $B_i \in \mathcal{A}$ , entonces  $A \in \mathcal{A}$ .

- Cambiar la gramática:
  - Eliminar producciones  $A \rightarrow \epsilon$
  - Producciones  $A \rightarrow \alpha B \beta$ , con  $B \in \mathcal{A}$  agregar  $A \rightarrow \alpha \beta$  (siempre que  $\alpha \beta \neq \epsilon$ )
  - Opcional: si realmente quiere mantener el mismo lenguaje, agregue  $S \rightarrow \epsilon$ . Más formal, si  $S \Rightarrow^* \epsilon$  ( $S \in \mathcal{A}$ ), agregar  $S \rightarrow \epsilon$ .

**Ejemplo:**

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aAA \mid \epsilon \\ B &\rightarrow bBB \mid \epsilon \end{aligned}$$

Primer paso: identificar no terminales que dan  $\epsilon$ :

$$\mathcal{A}^{(0)} = \{A, B\}$$

Dado que  $S$  nos lleva a  $A, B \in \mathcal{A}^{(0)}$ , también debemos agregarlo al conjunto  $\mathcal{A}^{(0)}$ :

$$\mathcal{A}^{(1)} = \{A, B, S\}$$

Nuevamente, debemos revisar algunos de los conjuntos presentes en  $\mathcal{A}^{(1)}$  nos llevan a producciones de la forma  $X \rightarrow \epsilon$ . Revisando nuevamente vemos que no hay más, por lo tanto:

$$\mathcal{A} = \{S, A, B\}$$

Cambiar la gramática:

$$S \rightarrow AB$$

$$A \rightarrow aAA$$

$$\cancel{A \rightarrow \epsilon}$$

$$B \rightarrow bBB$$

$$\cancel{B \rightarrow \epsilon}$$

$$S \rightarrow A \quad B \Rightarrow^* \epsilon$$

$$S \rightarrow B \quad A \Rightarrow^* \epsilon$$

$$A \rightarrow aA \quad A \Rightarrow^* \epsilon, \text{ se fue la primera } A \text{ o se fue la segunda } A \text{ en } A \rightarrow aAA$$

$$A \rightarrow a \quad A \Rightarrow^* \epsilon, \text{ se nos va la } A \text{ dos veces en } A \rightarrow aAA$$

$$B \rightarrow bB \quad B \Rightarrow^* \epsilon, \text{ se fue la primera } B \text{ o se fue la segunda } B \text{ en } B \rightarrow bBB$$

$$B \rightarrow b \quad B \Rightarrow^* \epsilon, \text{ se nos va la } B \text{ dos veces en } B \rightarrow bBB$$

$$S \rightarrow \epsilon \quad \text{Agregue esto sólo si quiere conservar el lenguaje original}$$

Pasos siguientes:

- Eliminar símbolos inútiles (no aparecen nunca en ninguna producción).

En  $G = (N, \Sigma, P, S)$  un símbolo  $X \in N \cup \Sigma$  se dice útil si hay una derivación  $S \Rightarrow^* \alpha X \beta \Rightarrow^* \sigma \in \Sigma^*$

- Propiedades de Clausura.

□

### 6.1.3. Forma Normal de Chomsky (CNF)

**Definición 6.3.** Una CFG  $G = (N, \Sigma, P, S)$  se dice en forma normal de Chomsky si todas sus producciones tienen las formas:

$$A \rightarrow BC \quad A, B, C \in N$$

$$A \rightarrow a \quad A \in N, a \in \Sigma$$

La forma normal de Chomsky (CNF) nos sirve para realizar demostraciones con gramáticas, ya que es mucho más sencillo si hay pocos tipos de lados derechos. /\* Tengo la duda en “si hay pocos tipos de lados derechos”... no entiendo esa parte. Considero que estaba mejor con “si sólo existe 1 árbol de derivación para la CFG.” \*/

**Teorema 6.1.** Sea  $G$  una CFG. Entonces puede construirse una CFG en CNF  $G'$  tal que  $\mathcal{L}(G) = \mathcal{L}(G')$ .

*Demostración.* Consideremos:

- Producciones:

$$A \rightarrow \alpha$$

con  $|\alpha| \geq 2$ , contiene terminales y no-terminales.

Inventar nuevos no-terminales  $A_a \rightarrow a$ , para todo  $a \in \Sigma$ , y reemplazar  $a$  por  $A_a$ .

- Producciones unitarias:  $A \rightarrow B$

Podemos identificar todos los no-terminales con producciones unitarias ( $A$ ) y sus producciones ( $A \rightarrow B$ ).

Para cada cadena,  $A \rightarrow B, B \rightarrow C, \dots, U \rightarrow V$  y  $V \rightarrow \alpha$ , con  $|\alpha| \geq 2$ , podemos reemplazar  $A \rightarrow \alpha, B \rightarrow \alpha, \dots$

- Producciones:

$$A \rightarrow B_1 B_2 \cdots B_n$$

Nuevos no-terminales  $X, \dots, Y$ :

$$\begin{aligned} A &\rightarrow B_1 X_1 \\ X_1 &\rightarrow B_2 X_2 \\ &\vdots \\ X_{n-2} &\rightarrow B_{n-1} B_n \end{aligned}$$

□

**Ejemplo 6.1** (Ejemplo CNF). Considerando la gramática del ejemplo 5.1:

$$E \rightarrow E + T \quad (1)$$

$$E \rightarrow T \quad (2)$$

$$T \rightarrow T * F \quad (3)$$

$$T \rightarrow F \quad (4)$$

$$F \rightarrow (E) \quad (5)$$

$$F \rightarrow a \quad (6)$$

**Fase 1:** Sólo  $A \rightarrow a$  o  $A \rightarrow B_1 B_2 \cdots B_n$  con  $B_i \in \mathbb{N}$ . Inventamos no-terminales:

$$M \rightarrow + \quad (7')$$

$$P \rightarrow * \quad (8')$$

$$A \rightarrow ( \quad (9')$$

$$C \rightarrow ) \quad (10')$$

y reemplazamos:

$$E \rightarrow EMT \quad (1')$$

$$E \rightarrow T \quad (2')$$

$$T \rightarrow TPF \quad (3')$$

$$T \rightarrow F \quad (4')$$

$$F \rightarrow AEC \quad (5')$$

$$F \rightarrow a \quad (6')$$

**Fase 2:** Eliminar producciones unitarias:

$$E \rightarrow T \quad (2')$$

$$T \rightarrow F \quad (4')$$

Para  $(4')$ , sabemos que las otras producciones son:

$$T \rightarrow TPF \quad (3')$$

$$T \rightarrow AEC \quad (4' \text{ y } 5')$$

$$T \rightarrow a \quad (4' \text{ y } 6')$$

Cadena:

$$\begin{aligned}
 & E \rightarrow T \Rightarrow \dots \\
 & \rightsquigarrow E \rightarrow TPF \quad (\text{reemplazando el lado derecho de 3'}) \\
 & \quad E \rightarrow AEC \\
 & \quad E \rightarrow a
 \end{aligned}$$

Por lo tanto, resulta:

$$\begin{aligned}
 & E \rightarrow EMT \\
 & \left. \begin{aligned} E &\rightarrow TPF \\ E &\rightarrow AEC \\ E &\rightarrow a \end{aligned} \right\} T \rightarrow F \\
 & T \rightarrow TPF \\
 & \left. \begin{aligned} T &\rightarrow AEC \\ T &\rightarrow a \end{aligned} \right\} T \rightarrow F \\
 & F \rightarrow AEC \\
 & F \rightarrow a \\
 & M \rightarrow + \\
 & P \rightarrow * \\
 & A \rightarrow ( \\
 & C \rightarrow )
 \end{aligned}$$

**Fase 3:** Producciones  $A \rightarrow B_1 \cdots B_n$ , con  $n \geq 3$  en CNE. Así:

$$\begin{aligned}
 & E \rightarrow EMT \\
 & \rightsquigarrow E \rightarrow EX_1 \\
 & \quad X_1 \rightarrow MT \\
 & E \rightarrow TPF \\
 & \rightsquigarrow E \rightarrow TX_2 \\
 & \quad X_2 \rightarrow PF \\
 & T \rightarrow TPF \\
 & \rightsquigarrow T \rightarrow TX_3 \\
 & \quad X_3 \rightarrow PF
 \end{aligned}$$

Finalmente, la forma normal de Chomsky de la gramática del ejemplo 5.1 es:

$$\begin{aligned}
 & M \rightarrow + \\
 & P \rightarrow * \\
 & A \rightarrow ( \\
 & C \rightarrow ) \\
 & E \rightarrow EX_1
 \end{aligned}$$

$$\begin{aligned}
X_1 &\rightarrow MT \\
E &\rightarrow TX_2 \\
X_2 &\rightarrow PF \\
T &\rightarrow TX_3 \\
X_3 &\rightarrow PF
\end{aligned}$$

## 6.2. Lema de Bombeo para Gramáticas de Contexto Libre

**Teorema 6.2** (Lema de Bombeo para CFL). Sea  $L$  una CFL. Entonces hay una constante  $n > 1$  tal que para todo  $\sigma \in L$ , si  $|\sigma| \geq n$  hay  $u, v, w, x, y \in \Sigma^*$  con los que podemos escribir:

$$\sigma = uvwxy$$

donde  $|vwx| \leq n$ ,  $vx \neq \epsilon$ , y para todo  $k \geq 0$ :

$$uv^kwx^ky \in L$$

*Demostración.* Sea  $L = \mathcal{L}(G)$ , con  $G = (N, \Sigma, P, S)$  una CFG en CNE. Consideremos un árbol de derivación de  $\sigma$  en  $G$  (Figura 6.1). Es importante destacar que el árbol de la Figura 6.1 corresponde a un

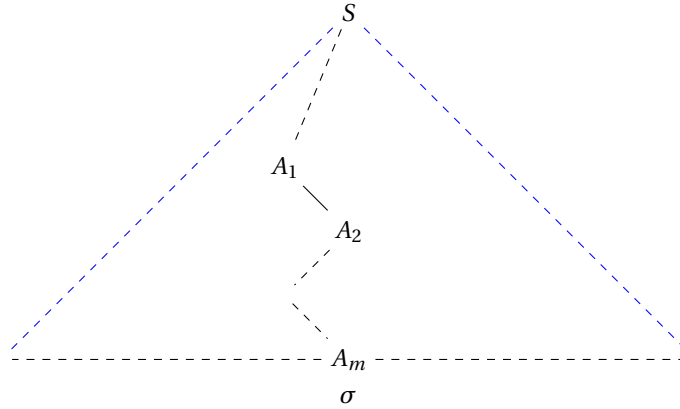


Figura 6.1: Para cada  $\sigma \in L$  tal que  $|\sigma| > N$ , existe un largo camino en el árbol de derivación. Nótese que este árbol tiene  $|\sigma|$  hojas.

árbol binario con  $|\sigma|$  hojas.

Sabemos que la altura del árbol (número de no-terminales en la rama más larga) crece con el número de hojas, correspondiente a  $|\sigma|$ . Si la altura del árbol es mayor que  $|N|$ , en la rama más larga un no-terminal se repite (Figura 6.2).

Como derivación, sabemos que hay  $A \in N$  tal que  $A \Rightarrow^+ vAx$ , y  $A \Rightarrow^+ w$  con  $vx \neq \epsilon$ ,  $v, w, x \in \Sigma^*$ .

$$S \Rightarrow^* uAy \Rightarrow^+ uvAxy \Rightarrow^* uvwxy$$

Pero entonces es una derivación:

$$S \Rightarrow^* uAy \Rightarrow^* uwy$$

y también:

$$\begin{aligned}
S &\Rightarrow^* uAy \Rightarrow^+ uvAxy \\
&\Rightarrow^+ uv^kAx^ky \\
&\Rightarrow^* uv^kwx^ky
\end{aligned}$$

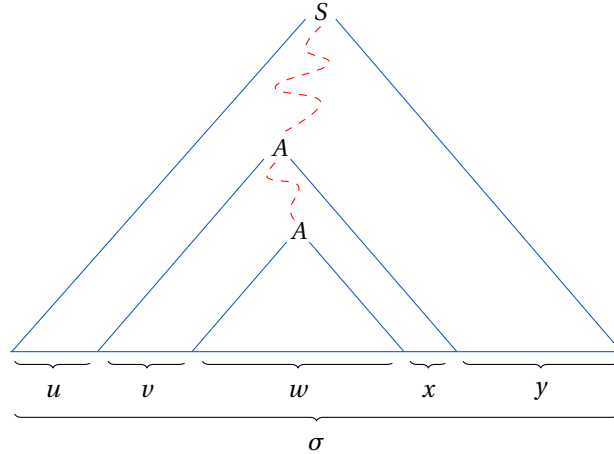


Figura 6.2: Por lema de bombeo para CFL, se puede descomponer  $\sigma = uvwxy$ .

O sea,  $uv^kwx^ky \in L$  para todo  $k$ .

Podemos aplicar la misma idea de muchas hojas  $\rightsquigarrow$  gran altura al árbol con raíz en  $A$ , y sus hojas ( $uvx$ ) no pueden ser demasiadas antes de que allí aparezca un no-terminal repetido. Esto da  $|vwx| \leq N$ .  $\square$

**Teorema 6.3.** El lenguaje

$$L = \{a^n b^n c^n : n \geq 1\} \quad (6.2.1)$$

no es CFL.

*Demostración.* Supongamos que  $L$  es CFL, con lo que cumple el lema de bombeo para CFL. Sea  $N$  la constante del lema. En seguida, consideremos  $\sigma = a^N b^N c^N$ ,  $\sigma \in L$ ,  $|\sigma| = 3N \geq N$ . Entonces, por PLCFL<sup>1</sup> hay  $u, v, w, x, y \in \Sigma^*$  tales que  $|vwx| \leq N$ ,  $vx \neq \epsilon$  tales que para todo  $k \geq 0$  se tiene que  $uv^kwx^ky \in L$ ; pero como  $|vwx| \leq N$ , está formado por a lo más dos tipos de símbolos. Al repetir  $vx$ ,

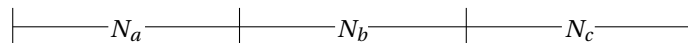


Figura 6.3: El lenguaje (6.2.1) exige que la cantidad de  $a$ 's sea igual a la cantidad de  $b$ 's y a la cantidad de  $c$ 's.

hay al menos un tipo de símbolo que no se repite, y ese queda corto. El resultado no pertenecen a  $L$ . Una contradicción.  $\square$

**Observación:** Un *lenguaje regular* es un caso particular de un CFL. Si demostramos que un lenguaje no es de *contexto libre*, se tiene que tampoco es regular. De la misma forma, si demostramos que un lenguaje es *regular*, entonces es de *contexto libre*.

### 6.3. Propiedades de Clausura para CFL

**Teorema 6.4.** Las CFL (context free language) son cerrados respecto operaciones regulares (unión, concatenación y estrella Kleene)

<sup>1</sup>Pumping Lemma for Context Free Language

**Ojo:** La estrella Kleene es complicada porque produce  $\epsilon$ ...

*Demostración.* Sean  $L = \mathcal{L}(G_1)$ ,  $L_2 = \mathcal{L}(G_2)$ , con  $G_1 = (N, \Sigma, P, S)$ ,  $G_2 = (N_2, \Sigma, P_2, S_2)$ .

**Unión** – Supongamos que  $N_1 \cap N_2 = \emptyset$ . Entonces generan:

$$L_1 \cup L_2 : G_u = (N_1 \cup N_2 \cup \{S\}, \Sigma, P_u, S)$$

$$\text{con } P_u = \{S \rightarrow S_1, S \rightarrow S_2\} \cup P_1 \cup P_2.$$

**Concatenación** – Para este caso tenemos:

$$L_1 \cdot L_2 : G_c = (N_1 \cup N_2 \cup \{S\}, \Sigma, P_c, S)$$

$$\text{con } P_c = \{S \rightarrow S_1 S_2\} \cup P_1 \cup P_2$$

**Estrella Kleene** – Se tiene:

$$L^* : G_a = (N_1 \cup \{S\}, \Sigma, P_a, S)$$

$$\text{con } P_a = \{S \rightarrow SS_1, S \rightarrow \epsilon\} \cup P_1.$$

□

**Teorema 6.5.** Los lenguajes de contexto libre **no** son cerrados respecto al complemento.

*Demostración.* Sabemos que  $L = \{a^n b^n c^n : n \geq 1\}$  no es CFL. Sin embargo,  $\bar{L}$  es CFL, ya que:

$$\bar{L} = \overline{\mathcal{L}(a^* b^* c^*)} \cup \{a^i b^j c^k : i \neq j \neq k\}$$

Los lenguajes regulares son cerrados respecto de su complemento, por lo tanto  $\overline{\mathcal{L}(a^* b^* c^*)}$  es regular y en consiguiente, de contexto libre.

El lenguaje de  $\{a^i b^j c^k : i \neq j \neq k\}$  es un lenguaje de contexto libre. Para poder construir su gramática debemos considerar el caso simple: construir una gramática para:

$$\{a^i b^j : i \neq j\} \tag{6.3.1}$$

El conjunto de producciones de (6.3.1) es:

$$S \rightarrow aSb \mid aAb \mid abB$$

$$A \rightarrow a \mid aA$$

$$B \rightarrow b \mid bB$$

De la misma forma, podemos construir  $\{a^i b^j c^k : i \neq j \neq k\}$  a través del siguiente conjunto de producciones:

$$\left. \begin{array}{l} S \rightarrow aSbC \mid aAb \\ A \rightarrow a \mid aA \\ C \rightarrow c \mid cC \end{array} \right\} \text{Permutaciones de esto dan la segunda parte}$$

Como existe una gramática de contexto libre para  $\{a^i b^j c^k : i \neq j \neq k\}$ , entonces puede generar un lenguaje de contexto libre (valga la redundancia).

Los lenguajes de contexto libre son cerrados respecto a la unión, por ende,  $\bar{L}$  también es de contexto libre. □

**Importante:** Cuando decimos que un conjunto de lenguajes no es cerrado respecto a una operación  $x$ , no quiere decir que siempre al aplicar  $x$  sobre un lenguaje del conjunto vamos a tener un conjunto que no es de contexto libre... tenga cuidado.

Consideremos el hecho de los lenguajes regulares. Sabemos que el complemento de un lenguaje regular sigue siendo regular. Según la Jerarquía de Chomsky, se tiene que los lenguajes regulares también son de contexto libre, por lo tanto, el complemento de **ESE** caso particular CFL sí da de contexto libre.

**Teorema 6.6.** CFL son cerrados respecto a la intersección con lenguajes regulares.

*Demostración.* Similar a la intersección entre lenguajes regulares:

- $L_1 = \mathcal{L}(M_1)$ ,  $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_1, Z_1, F_1)$  un PDA.
- $L_2 = \mathcal{L}(M_2)$ ,  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$  un NFA.

La idea es construir un PDA  $M$  que siga la pista de  $M_1$  y  $M_2$  en componentes del estado

$$M = (Q_1 \times Q_2, \Sigma, \Gamma, \delta, (q_1, q_2), Z_1, F_1 \times F_2)$$

donde construimos

$$\delta((q', q''), X, A) = \{((p', p''), \gamma) : (p', \gamma) \in \delta_1(q', X, A) \wedge p'' \in \delta_2(q'', X)\}$$

Entonces [furioso agitar de brazos]  $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$  □

**Teorema 6.7.** Los CFL son cerrados respecto substituciones.

*Demostración.* Sea  $L$  un CFL sobre  $\Sigma$ , y sea  $L_a$  sobre  $\Delta$  un CFL para todo  $a \in \Sigma$ . Supongamos que  $L = \mathcal{L}(G)$ , con  $G = (N, \Sigma, P, S)$  y  $L_a = \mathcal{L}(G_a)$ , con  $G_a = (N_a, \Delta, P_a, S_a)$ , ambos CFG's. Sin pérdida de generalidad, los conjuntos  $N$ ,  $N_a$  son disjuntos a pares (no hay terminales en común entre gramáticas). La gramática que resulta de  $G$  substituyendo  $a \mapsto S_a$  (donde cada  $S_a$  es CFL), y agregando todas las producciones de las  $G_a$ , genera la substitución pedida. □

**Corolario 6.7.1.** CFL son cerrados respecto al homomorfismo.

## 6.4. Ejercicios

---

**Ejercicio 6.1.** Demuestre que el lenguaje

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

no es regular.

**Ejercicio 6.2.** Sea  $G$  en CNE,  $\sigma \in \mathcal{L}(G)$ . Demuestre que la derivación de  $\sigma$  toma  $2|\sigma| - 1$  pasos.



## **Parte II**

# **Computabilidad y Problemas Intratables**



## Capítulo 7

# Computabilidad

En este capítulo cambiaremos de dirección drásticamente. Desde ahora, estaremos interesados primeramente clases simples de lenguajes (programas) y la manera en que ellos pueden ser utilizados para problemas relativamente limitados, tales como analizadores de protocolos, búsqueda de texto o analizadores sintácticos. Comenzaremos en la búsqueda de qué lenguajes pueden ser definidos por cualquier modelo computacional. Esta pregunta es equivalente a la pregunta “¿Qué es computación?, ¿Qué puede o no hacer?”.

Para ello, introduciremos de manera informal al concepto de “no-decibilidad”, que en el fondo son problemas que una computadora no puede resolver. En seguida, la formalizaremos al introducir un nuevo modelo de computación: las máquinas de Turing.

### 7.1. Problemas que los computadores no pueden resolver

---

El propósito de esta sección es entregar, al mejor estilo de programación en C, una introducción para demostrar un problema específico que las computadoras no pueden resolver. El problema particular que discutiremos es si un programa en C imprime `Hola Mundo` como primera cosa.

#### 7.1.1. El hipotético detector de “Hola Mundo”

Consideremos el clásico programa que hasta un mechón puede hacer:

```
#include <stdio.h>
main()
{
    printf("Hola, mundo!\n");
}
```

Pregunta: ¿Podemos escribir un programa que detecte “Hola, mundo!”? (dado un programa, determina si lo primero que escribe es “Hola, mundo”)

*Demostración.* Consideremos los programas en “C”, con verdaderos enteros como `int`; sin reales. Buscamos construir un programa  $H$  (programa que detecta si **otro** programa escribe “Hola, mundo”) (Figura 7.1). Los programas leen de entrada estándar únicamente y escriben a salida estándar exclusivamente.

Primer cambio: Modificamos  $H$  de manera que lea sólo el programa, y use el texto del programa como datos. Lo llamaremos  $H'$  (Figura 7.2).

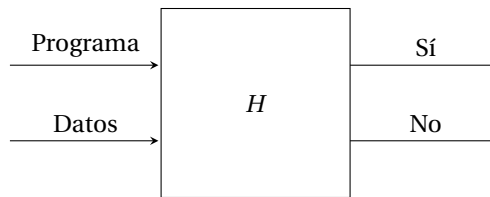


Figura 7.1: Si el programa de entrada “Programa” escribe en algún momento “Hola, mundo”, la salida de  $H$  será “Sí”. En caso contrario será “No”. Nótese que estamos suponiendo que  $H$  está escrito en C y usa su biblioteca estándar.

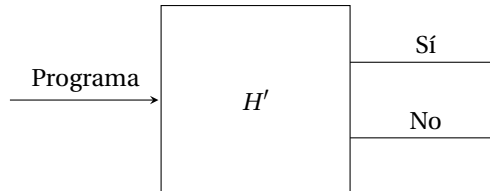


Figura 7.2: El texto que “Programa” utiliza como datos se guarda previamente en memoria antes de su ejecución.

Segunda Modificación: Modificamos  $H'$  de manera que en vez de escribir “No”, escribe “Hola mundo”. El resultado es  $H''$  (Figura 7.3).

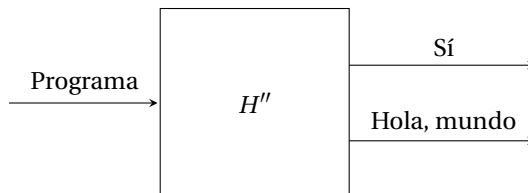


Figura 7.3: Cuando  $H''$  detecta que “Programa” escribe “Hola, mundo”, en lugar de arrojar como salida un “No”, saldrá un “Hola, mundo”.

¿Qué pasa si alimentamos  $H''$  con  $H''$ ? **Hint:** Suponga que  $H''$  es un archivo .c gigantesco (no modularizado)

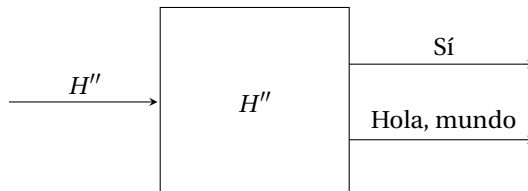


Figura 7.4: Independientemente de lo que lea  $H''$  siempre habrá una contradicción.

Ocurre lo siguiente:

- Si  $H''$  leyendo  $H''$  responde “Sí”, debiera escribir “Hola, mundo”. Es una contradicción.
- Si  $H''$  leyendo  $H''$  responde “Hola mundo”, debiera escribir “Sí”. Es una contradicción.

Por lo tanto,  $H$  no existe.

□

**Observación:** Nótese que cada vez que construimos una modificación del programa  $H$  o  $H'$ , el siguiente programa sólo es subconjunto de lo anterior (Figura 7.5)

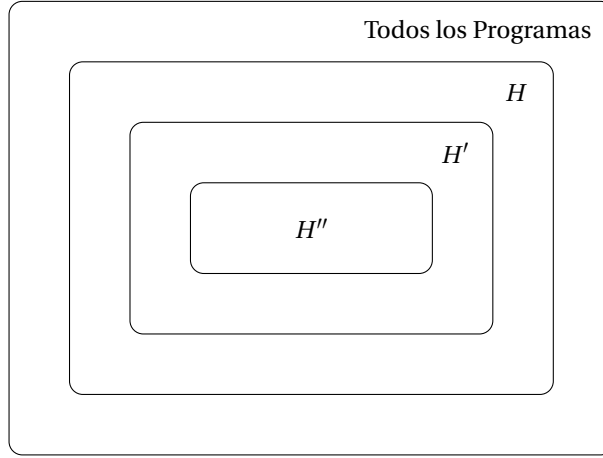


Figura 7.5: Si  $H'$  no funciona, entonces  $H'$  y  $H$  tampoco. Esto funciona de la misma manera que los lenguajes regulares y las gramáticas de contexto libre (Si no es regular, entonces no es de contexto libre)

### 7.1.2. Reducir un problema a otro

**Definición 7.1.** Un *problema*  $P$  es un lenguaje infinito sobre  $\Sigma$ . Resolver el problema es construir un artilugio  $S$  que reconozca  $P$ . O sea, dado  $\sigma \in \Sigma^*$ ,

$$S(\sigma) = \begin{cases} \text{Verdadero,} & \sigma \in P \\ \text{Falso,} & \sigma \notin P \end{cases}$$

**Definición 7.2** (Decidible). Informalmente, un programa  $P$  es decidible si dado una entrada  $\sigma \in \Sigma^*$  es capaz de responder “sí” o “no”.

**Observación:** Si limitamos  $\sigma$  finito, entonces sólo basta con hacer una tabla que abarque todos los casos posibles de  $\sigma$  cuando el problema es Verdadero, o es Falso (Tabla de resultados pre-calculados).

**Definición 7.3.** Un problema  $P$  se reduce al problema  $Q$  si hay un algoritmo (procedimiento finito) que dado  $\sigma \in \Sigma$  entrega  $\sigma''$  tal que:

$$\sigma \in P \Leftrightarrow \sigma' \in Q$$

Se anota  $P \leq Q$  ( $P$  es más fácil que  $Q$ ). Dicho de otra forma, lo que hacemos es transformar el problema a un ámbito distinto que resuelve el mismo problema. Por ejemplo, recuerde en MAT023 cuando resolvíamos EDO's con transformada de Laplace (Figura 7.6).

**Observación:** Si  $P$  no tiene solución,  $Q$  tampoco ( $Q$  es a lo menos tan difícil como  $P$ ).

**Ejemplo 7.1.** Determinar si en un programa alguna vez se le asigna un valor a la variable  $x$ .

*Demostración.* La idea consiste en reducir el problema del detector de “Hola Mundo” a esto. Para ello definimos un programa  $P$  que imprime “Hola mundo” como primera cosa. Sea  $Q$  otro programa que asigna un valor a la variable  $x$ . Luego, hacemos la reducción  $P \leq Q$  a través de:

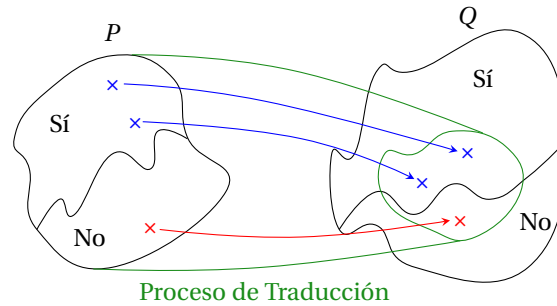


Figura 7.6: El proceso de transformación traduce el problema  $P$  a un problema  $Q$  que es más difícil.

- 1) Sea  $G$  otro programa que toma cada `printf("Hola mundo")` y lo transforma a una asignación de la variable  $x$ , es decir:

`printf("Hola mundo")  $\rightsquigarrow$  x = <algo>`

La Figura 7.7 describe usando un diagrama de cajitas y flechas que es lo que está ocurriendo. Cabe destacar, que el programa  $G$  debe evitar éxitos accidentales, es decir, que  $P$  asigne  $x$  en

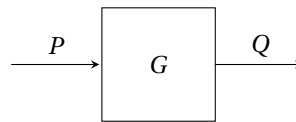


Figura 7.7: El programa  $G$  transforma cada `printf("Hola mundo")` en una asignación de la variable  $x$ . En el fondo es  $P \leq Q$ .

algún momento. Para ello, debemos cambiar todas las asignaciones de la variable  $x$  en  $P$  por cualquier otra variable. Por ejemplo:

`x = algo  $\rightsquigarrow$  dummy_x = algo`

- 2) Sea  $H$  un programa decidible que detecta si otro programa le asigna un valor a la variable  $x$ . Luego, le entregamos la salida de  $G$  a  $H$ . La Figura 7.8 muestra dicha construcción.

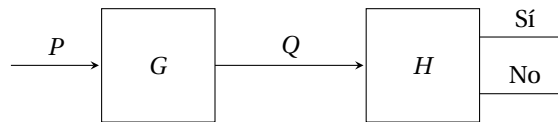


Figura 7.8: El programa  $G$  transforma cada `printf("Hola mundo")` en una asignación de la variable  $x$ . En el fondo es  $P \leq Q$ .

- 3) Sea  $H'$  el programa que encapsule toda la construcción de la Figura 7.8 (Figura 7.9). Como

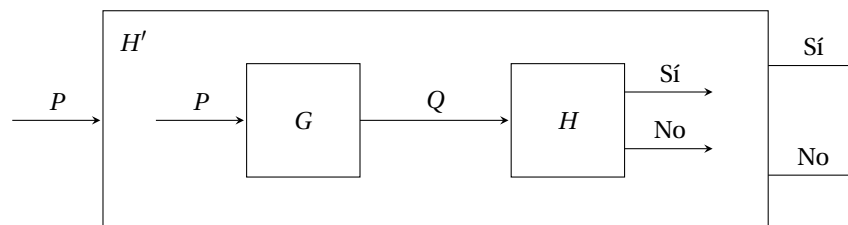


Figura 7.9: La construcción de la Figura 7.8 puede encapsularse en otro programa  $H'$ .

puede observar,  $H'$  no es más que un detector de “Hola mundo”, problema que sabemos de por sí que no es decidible, en consecuencia,  $H$  no puede existir.

□

Algunas variantes de este programa:

- Detectar si un programa define una función  $f$ .
- Detectar si dos funciones  $g$  y  $h$  de un programa imprimen la misma cosa.

## 7.2. Máquinas de Turing

**Definición 7.4.** Una *máquina de Turing* (TM) consta de:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F) \quad (7.2.1)$$

Donde:

- $Q$ : conjunto finito de estados.
- $\Sigma$ : alfabeto de entrada.
- $\Gamma$ : alfabeto de la cinta.
- $\delta$ : función de transición.
- $q_0 \in Q$ : estado inicial.
- $B$ : Blanco  $B \in \Gamma$ ,  $B \notin \Sigma$ . Considere estos símbolos como “auxiliares”.
- $F \subseteq Q$ : Estados finales.
- $\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, D\}}$ , donde  $L$  es movimiento del cabezal 1 posición hacia la izquierda, y  $D$  hacia la derecha.

**Importante:** Esto es un NFA.

**Idea:**  $M$  trabaja sobre una *cinta* ( $\rightsquigarrow$  string en  $\Gamma$ ), si  $M$  está viendo el símbolo  $x$  y está en el estado  $q$ , en un paso o cambiar el estado a  $p$  y sobrescribe  $x$  con  $y$ , y luego se mueve hacia la izquierda o derecha, dependiendo si  $\delta(q, x) \ni (p, y, L)$  o  $\delta(q, x) \ni (p, y, D)$  respectivamente.

Algunas observaciones de las máquinas de Turing:

- Las máquinas de Turing sólo aceptan por estado final.
- Toda idea de computación, se ha demostrado que es equivalente a una máquina de Turing, en otras palabras, podemos representar a través de una TM un DFA, NFA, PDA, etc.

### 7.2.1. Técnicas de Programación para Máquinas de Turing

**Importante:** El cabezal de una TM puede mantenerse fijo. Para lograrlo, se debe hacer una jugarreta de movidas.

A continuación, mostraremos algunos ejemplos que darán a conocer las habilidades de la TM. Nótese que estas ideas no extienden el modelo básico de la TM; sólo son convenciones en cuanto a su notación.

- **TM de Múltiples Pistas** – Simple, considere que  $\Gamma$  es un conjunto de objetos  $[x, a_1, a_2, \dots, a_n]$ , donde  $x \in \Sigma$  y  $a_1, \dots, a_n$  son los símbolos en las demás pistas (Figura 7.10).

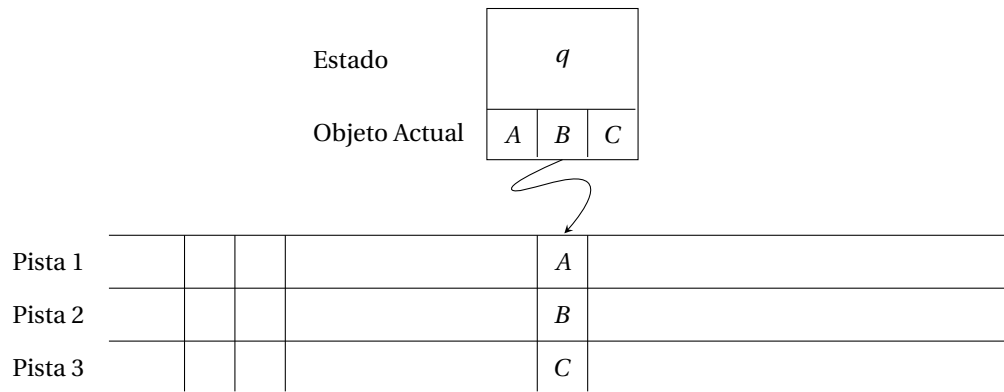


Figura 7.10: Máquina de Turing de múltiples pistas, cuyo cabezal corresponde al vector  $[A, B, C]$

- **TM de Múltiples Cabezales** – Podemos simularlo con múltiples pistas (Figura 7.11). Una pista contiene el contenido de la cinta. Cabe destacar que esta TM es no-determinista.

							*	
	*							
			*					
$a$	$b$	$a \dots$	$c$	$\dots$	$d$	$\dots$	$a$	

Figura 7.11: Los \* indican la posición de uno de los cabezales.

- **TM de Múltiples Cintas** – Se simula mediante dos pistas por cinta: contenido y posición del cabezal.
- **TM de cinta infinita en ambas direcciones** – La simulación se hace colocando un símbolo \$ en algún lugar de la cinta, tal que la computación sobre la TM comience a través de él (Figura 7.12).

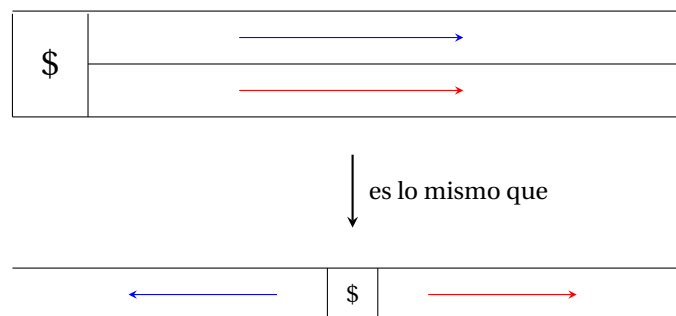


Figura 7.12: Simulación de una máquina de Turing de cinta infinita en ambas direcciones.

### 7.2.2. Simular un Procesador

Lenguaje “de máquina”:

- Registros:  $R_0, R_1, \dots$



## ■ Operaciones:

```
     $R_i \leftarrow \text{cte}$   
     $R_i \leftarrow \text{mem}[R_j]$   
     $R_i \leftarrow R_j$   
     $\text{mem}[R_j] \leftarrow R_i$   
     $\vdots$   
    goto posicion  
    if  $R_i == 0$  goto posición
```

## ■ Memoria



## Capítulo 8

# Indecibilidad

### 8.1. Máquina de Turing Universal

---

**Idea:** Construir una máquina de Turing universal tal que  $U$  acepta  $\langle M, \sigma \rangle$ , si la TM  $M$  acepta  $\sigma$  ( $M, \sigma$  representada en forma “cuerda”) (esta máquina de Turing acepta siempre y cuando  $M$  acepte  $\sigma$ ).

Supongamos que la TM  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ . Podemos suponer:

1. Hay un único estado final.
2.  $\Sigma = \{0, 1\}$

Para los puntos (2) y (3): Si “realmente” necesitamos, por ejemplo,  $\Gamma$  un alfabeto mayor, podemos codificar los símbolos en varios bits, y antes de comenzar el proceso “traducir” lo que viene en la cinta.

3.  $\Gamma = \{0, 1, B\}$
4. Estados son  $\{q_1, q_2, q_3, \dots, q_r\}$ ;  $q_1$  es inicial,  $q_2$  es final.
5. Símbolos en  $\Gamma$  se numeran:

- $0 \rightarrow 1$
- $1 \rightarrow 2$
- $B \rightarrow 3$

6. Direcciones:

- Izquierda  $\rightarrow 1$
- Derecha  $\rightarrow 2$

7. Podemos representar la función  $\delta$ :

$$\delta(q, a) = \{(p_1, a_1, d_1), (p_2, a_2, d_2), \dots\}$$

Con  $q \in Q$ ,  $a \in \Gamma$ ,  $p_i \in Q$ ,  $a_i \in \Gamma$  y  $d_i \in \{\text{izq}, \text{der}\}$

8. En vez de escribir el conjunto damos cada posibilidad, o sea listar los quintetos:

$$q, a, p_i, a_i, d_i$$

Estos podemos representarlos:

$$0^q 10^a 10^{p_i} 10^{a_i} 10^{d_i} \tag{8.1.1}$$

Para representar  $M$ , listamos los quintetos de todos sus movidas posibles, separados por 11.

Una secuencia de  $\{0, 1\}$  que no tenga el “formato correcto” (no comienza en 0, no están en  $\mathcal{L}((0^+10^+10^+10^+11)^*0^+10^+10^+10^+10^+)$ ) consideramos que representa la TM de la Figura 8.1.



Figura 8.1: Máquina de Turing que no cumple con el formato  $\mathcal{L}((0^+10^+10^+10^+11)^*0^+10^+10^+10^+10^+)$

$\rightsquigarrow$  Tenemos una función de  $\{0, 1\}^* \rightarrow \text{TM}$ .

Podemos numerar los elementos de  $\{0, 1\}^*$ :

$\epsilon \rightarrow 1$   
 $0 \rightarrow 2$   
 $1 \rightarrow 3$   
 $00 \rightarrow 4$   
 $01 \rightarrow 5$   
 $10 \rightarrow 6$   
 $11 \rightarrow 7$   
 $\vdots$

Esto es una biyección entre  $\mathbb{N}$  y  $\Sigma^*$ .

Interesa representar  $\langle M, \sigma \rangle$ , donde  $M$  es una TM,  $\sigma \in \{0, 1\}^*$  son datos de entrada. Una opción es:

$$\langle M \rangle 111\sigma \tag{8.1.2}$$

En otras palabras, (8.1.2) nos dice que separamos la máquina de Turing  $M$  de los datos de entrada  $\sigma$  a través de una secuencia de tres 1's. Por otro lado, la relación (8.1.2) me permite hablar de la TM número  $i$ ,  $M_i$ , y los datos de entrada número  $j$ ,  $\sigma_j$ .

### 8.1.1. El lenguaje diagonal

**Definición 8.1.** Definimos el lenguaje diagonal  $L_d$ , como:

$$L_d = \{i : \sigma_i \notin \mathcal{L}(M_i)\} \tag{8.1.3}$$

Con dibujitos (Cuadro 8.1):

		$\sigma_i \rightarrow$				
		1	2	3	4	5
$M_i \downarrow$	1	1	0	1	...	...
	2	0	0	1	...	...
	3	1	0	0	...	...
	4	$\vdots$	$\vdots$	$\vdots$	1	...
	5	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Cuadro 8.1: Los números rojos representan la diagonal de la matriz.

Observaciones importantes que acotar de la matriz (Cuadro 8.1):

- Si  $a_{ij} = 0$ , entonces la Máquina de Turing  $M_i$  no acepta  $\sigma_j$ .

- Si  $a_{ij} = 1$ , entonces la Máquina de Turing  $M_i$  acepta  $\sigma_j$ .

¿Hay una máquina de Turing que acepte  $L_d$ ?

**Teorema 8.1.** No hay máquina de turing que acepte  $L_d$ .

*Demostración.* Por contradicción. Supongamos que  $M$  acepta  $L_d$ . Entonces  $M = M_i$  para algún  $i$ . Entonces  $M_i$  acepta  $\sigma_i$ , pero eso es que  $i \notin L_d$ . Lo cual es una contradicción.  $\square$

$L_d$  en el fondo son máquinas de turing que no se aceptan a si mismas  $\rightarrow$  problema de “Hola mundo”.

La demostración del Teorema 8.1 era algo bastante esperado, ya que hay más lenguajes que máquinas de turing, por lo tanto, hay lenguajes que no pueden ser representados por TM.<sup>1</sup>

**Observación** Nótese que  $L_d$  al ser *no decidible*, se tiene que tampoco es *recursivamente enumerable*.

**Definición 8.2** (Recursivamente Enumerable). Un lenguaje se llama *recursivamente enumerable* (R.E.) si hay una TM que lo acepte.

**Definición 8.3** (Recursivo). Hay un algoritmo (instrucciones precisas, siempre termina). En términos de una máquina de Turing, un lenguaje recursivo es una máquina de Turing que siempre se detiene respondiendo “sí” o “no” (decible).

Algunas observaciones de las definiciones 8.2 y 8.3:

- Si un lenguaje es recursivo es decidible.
- Un lenguaje no-decidible no es recursivo

**Teorema 8.2.**  $L_d$  no es R.E.

*Demostración.* Por contradicción. Supongamos que  $L_d = \mathcal{L}(M_k)$ . Consideremos  $\sigma_k$ . ¿Pertenece o no a  $L_d$ ?

- Si  $\sigma_k \in L_d$ , quiere decir que  $\sigma_k \in \mathcal{L}(M_k)$ , es una contradicción.
- Si  $\sigma_k \notin L_d$ , quiere decir que  $\sigma_k \notin \mathcal{L}(M_k)$ , o sea,  $\sigma_k \in L_d$ . Una contradicción.

$\square$

**Teorema 8.3.** Si  $L$  es aceptado por una TM, es aceptado por una TM determinista.

*Demostración.* Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  una TM. Podemos suponer sin pérdida de generalidad que  $Q \cap \Gamma = \emptyset$  (queremos anotar IDs en una cinta), y elegimos un separador  $\# \notin Q \cup \Gamma$ .

Nuestra idea es escribir IDs de  $M$ , separados por  $\#$ , en la cinta. Construimos una TM que hace lo siguiente:

- Dado  $\sigma$ , anota  $\#q_0\sigma\#$  en su cinta. ( $q_0\sigma$  corresponde a una descripción instantánea de la TM)
- En un ciclo:
  - Retrocede al último ID no cerrado en la cinta.
  - Dado el estado en el ID con el símbolo siguiente, determinar las movidas posibles de  $M$ .  
Para cada movida de  $M$ :
    - Copia el último ID al final, modificandolo para indicar esa movida.

<sup>1</sup>Para entender de mejor manera esto, ver el capítulo de “Numerabilidad” del apunte de Fundamentos de Informática

- Si el ID construido recién es de aceptación, acepta.
- Cierra el ID considerado.

Por ejemplo, si en una descripción instantánea  $ID_0$  de la TM vemos que tiene tres movidas posibles, entonces se van a originar tres descripciones instantáneas diferentes ( $ID_1$ ,  $ID_2$  e  $ID_3$ ). De la misma forma, si  $ID_1$  tiene dos movidas posibles, se van a originar 2 descripciones adicionales ( $ID_4$  e  $ID_5$ ).

Luego, el esquema de la idea es:

- Comenzamos con la  $ID_0$ :

$\#ID_0\#$

- $ID_0$  tiene tres movidas posibles, representadas por  $ID_1$ ,  $ID_2$  e  $ID_3$ . Se agregan a la cinta:

$\#ID_0\#ID_1\#ID_2\#ID_3\#$

- Ya agregamos todas las descripciones instantáneas de  $ID_0$ , así que ahora hacemos lo mismo con  $ID_1$ .

Previamente dijimos que presenta dos opciones representadas por las descripciones instantáneas  $ID_4$  e  $ID_5$ . Las agregamos:

$\#ID_0\#ID_1\#ID_2\#ID_3\#ID_4\#ID_5\#$

- Ya agregamos todas las descripciones instantáneas de  $ID_1$ , así que ahora continuamos agregando las descripciones instantáneas de  $ID_2$ .

$\vdots$

□

**Teorema 8.4.**  $\bar{L}_d$  es recursivamente enumerable.

*Demostración (vía concepto de enumerar).*  $\bar{L}_d = \{\sigma_i : \sigma_i \in \mathcal{L}(M_i)\}$

Podemos **enumerar**  $\bar{L}_d$  vía:

- Ejecute  $M_1$  sobre  $\sigma_1$  por 0 pasos.
- Ejecute  $M_1$  sobre  $\sigma_1$  por 1 pasos.
- Ejecute  $M_2$  sobre  $\sigma_2$  por 0 pasos.
- Ejecute  $M_1$  sobre  $\sigma_1$  por 1 pasos.
- Ejecute  $M_2$  sobre  $\sigma_2$  por 1 pasos.
- Ejecute  $M_i$  sobre  $\sigma_i$  por  $k$  pasos (en orden de  $i + k$  creciente)

En el fondo lo que estamos haciendo es lo siguiente:

□

*Demostración (vía definición de  $\bar{L}_d$ ).* Alternativamente, dado  $i$ , simule  $M_i$  sobre  $\sigma_i$ ; por definición de  $\bar{L}_d$ , en algún momento tendrá que aceptar. □

**Teorema 8.5.** Si  $L$  es recursivamente enumerable, y su complemento es recursivamente enumerable, entonces  $L$  es recursivo (y  $\bar{L}$  también).

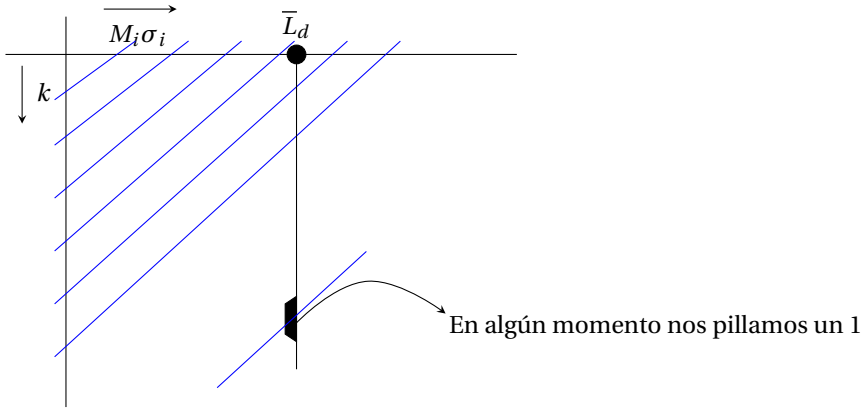


Figura 8.2: Cada corte diagonal representa ejecutar la Máquina de Turing  $M_i$  con entrada  $\sigma_i$  en  $k$  pasos.

*Demostración.* Suponemos dadas TM  $M$  con  $L = \mathcal{L}(M)$  y  $\overline{M}$  con  $\overline{L} = \mathcal{L}(\overline{M})$ .

Corremos  $M$  y  $\overline{M}$  en paralelo, la primera que acepte nos dice si  $\sigma \in L$  o  $\sigma \in \overline{L}$ , con lo que tenemos que una TM que siempre se detiene que acepta  $L$ . O sea,  $L$  es recursivo (y también lo es  $\overline{L}$ )  $\square$

**Corolario 8.5.1.** Si  $L$  es R.E., pero no recursivo,  $\overline{L}$  no es R.E.

Tablita de posibilidades:

$L$ recursivo	y	$\overline{L}$ recursivo
$L$ R.E.	y	$\overline{L}$ R.E. $\Rightarrow L, \overline{L}$ recursivos.
$L$ R.E., no recursivo	y	$\overline{L}$ no R.E.
$L$ no R.E.	y	$\overline{L}$ no R.E.

Cuadro 8.2: Tablita de posibilidades del Teorema 8.5. Nótese que los dos primeros casos son exactamente el mismo.

## 8.2. Demostración de lenguajes recursivos y R.E. usando cajitas

**Idea** Representar TM por una cajita.

- Si  $M$  es una TM que siempre se detiene (Figura 8.3).

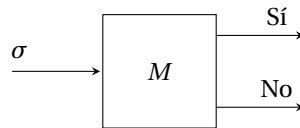


Figura 8.3: Si  $M$  es capaz de responder “Sí” o “No” (decidible) ante una entrada  $\sigma$ , entonces es recursivo. En otras palabras,  $\mathcal{L}(M)$  es recursivo.

- Si  $M$  es una TM que genera un lenguaje recursivamente enumerable (Figura 8.4)

Casos como los de la Figura 8.4 puede ocurrir: la TM se queda dando vueltas indefinidamente hasta dar una respuesta.

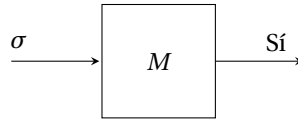


Figura 8.4: Si  $M$  acepta una entrada  $\sigma$ , entonces es recursivamente enumerable. Esto es lo mismo que  $\mathcal{L}(M)$  es R.E.

**Teorema 8.6.** Si  $L$  es recursivo,  $\bar{L}$  es recursivo.

*Demostración.* Sea  $L = \mathcal{L}(M)$ . La TM de la Figura 8.5 acepta  $\bar{L}$ . □

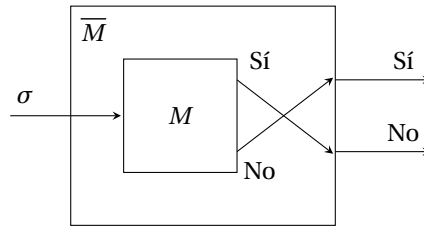


Figura 8.5: Construcción de  $\bar{M}$ . Como puede observar, si  $M$  acepta, entonces  $\bar{M}$  no acepta y viceversa.

**Teorema 8.7.** Si  $L$  y  $\bar{L}$  son R.E., ambos son recursivos. *Dicho de otra forma, si sabemos que  $L$  y  $\bar{L}$  son recursivamente enumerables, entonces  $L$  y  $\bar{L}$  son recursivos (recuerde que recursivo es más fuerte que R.E.).*

*Demostración.* Sea  $L = \mathcal{L}(M)$ ,  $\bar{L} = \mathcal{L}(\bar{M})$ . La TM de la Figura 8.6 acepta  $L$ .

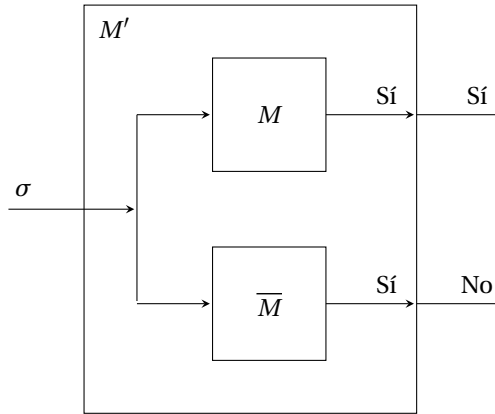


Figura 8.6: Máquina de Turing que echa a correr  $\sigma$  sobre  $L$  y  $\bar{L}$  en paralelo.

Echamos a correr  $M$  y  $\bar{M}$  sobre una palabra  $\sigma$  en paralelo (cada vez que una TM haga una movida, la otra también hará su propio movimiento). alguna de las dos TM tendrá que responder primero, ya que si  $\sigma \notin L$ , entonces,  $\sigma \in \bar{L}$  y viceversa. Luego, integramos  $M$  y  $\bar{M}$  en una TM  $M'$  que, se encarga de transformar el "Sí" de  $\bar{M}$  en un "No" de  $M'$ . Dado esto, vemos que si  $L$  y  $\bar{L}$  son R.E., entonces podemos construir una TM que nos diga que ambos lenguajes en realidad son recursivos. □

Usando cajitas podemos demostrar que:



- Demostrar que la unión de dos lenguajes recursivos también es recursivo.
- Cerrado respecto al complemento: use el complemento y la unión.
- Se sospecha que son cerrados respecto a las operaciones de las expresiones regulares.

Algunas observaciones:

- Aceptado no significa que siempre va a tener la respuesta: Recuerde los casos donde la máquina se queda dando vueltas, y nunca entrega la respuesta pero si acepta la palabra que se le ingrese.
- $L_u$  toma una máquina de Turing y una palabra  $\sigma$

### 8.3. Reducción de Problemas

**Definición 8.4** (Problema). Lenguaje  $\subseteq \Sigma^*$ , determinar si  $\sigma$  pertenece o no. Interesan lenguajes infinitos (no tiene gracia una simple lista finita).

**Definición 8.5** (Reducción). Algoritmo que transforma  $\alpha \in \Sigma^*$  (el problema es ¿es  $\alpha \in A$ ?) en  $\beta \in \Gamma^*$  (problema es ¿es  $\beta \in B$ ?) tal que  $\alpha \in A \Leftrightarrow \beta \in B$ . Si hay una reducción se anota  $A \leq B$  (Figura 8.7).

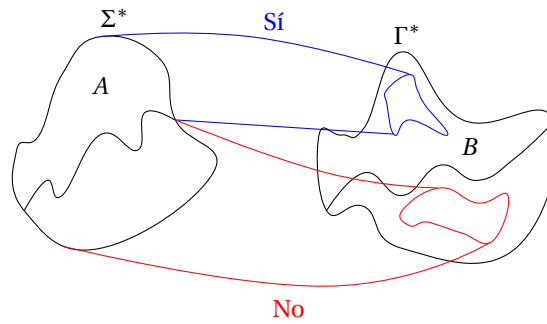


Figura 8.7: Representación gráfica de la reducción de problemas. En este caso, reducimos el problema  $A$  a instancias del problema  $B$ , donde  $B$  es a lo menos, tan difícil como  $A$ .

**Teorema 8.8.** Si  $P_1 \leq P_2$ , entonces:

- Si  $P_1$  no es decidible,  $P_2$  no es decidible.
- Si  $P_1$  no es recursivamente enumerable,  $P_2$  no es recursivamente enumerable.

**Demostración.** Por contradicción. Ejercicio: realmente entender esto. □

Un par de lenguajes útiles:

$$\left. \begin{array}{l} L_e = \{M : \mathcal{L}(M) = \emptyset\} \\ L_{ne} = \{M : \mathcal{L}(M) \neq \emptyset\} \end{array} \right\} \text{ Notar que } L_c = \overline{L_{ne}} \text{ (} e \rightsquigarrow \text{empty, } ne \rightsquigarrow \text{not empty)}$$

**Teorema 8.9.**  $L_{ne}$  es R.E.

**Demostración.** Usamos  $L_u$  (también es R.E.) (lenguaje aceptado por una máquina de Turing universal) en nuestra construcción (Figura 8.8). □

**Teorema 8.10.**  $L_{ne}$  no es recursivo (=  $L_e$  no es recursivamente enumerable)

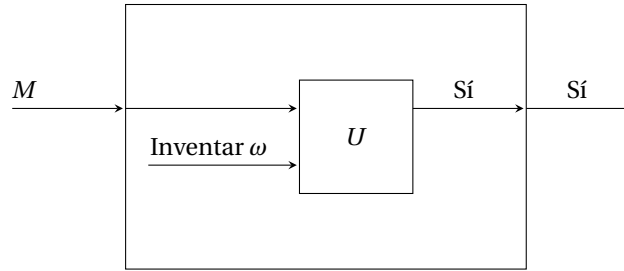


Figura 8.8: Inventamos una palabra  $\omega$  tal que  $U$  la acepte. Nótese que este  $\omega$  inventado representa nuestro no-determinismo.

**Demostración.** Reducimos  $L_u$  a  $L_{ne}$  (la reducción de problemas también se aplica para deducir si un lenguaje es recursivamente enumerable o no). Sabemos que  $L_u$  es R.E., no recursivo

En castellano, dado  $\langle M, \omega \rangle$  en  $L_u$ , debo construir una TM que acepte el lenguaje vacío si  $M$  no acepta  $\omega$  y un lenguaje no vacío (por ejemplo  $\Sigma^*$ ) si  $M$  acepta  $\omega$ . Dados  $M$  y  $\omega$  puedo construir  $M'$  (Figura 8.9).

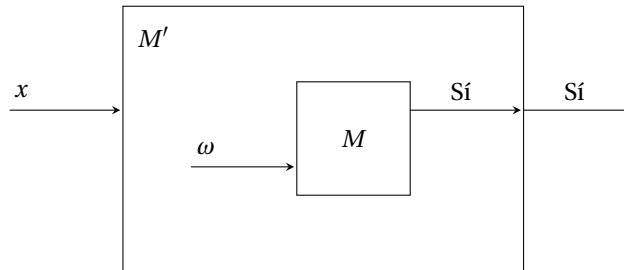


Figura 8.9: /\* Agregar alguna leyenda \*/

Esto es  $L_u \leq L_{ne}$ ;  $L_u$  no recursivo  $\Rightarrow L_{ne}$  no recursivo.

□

## 8.4. Teorema de Rice

**Definición 8.6.** Una *propiedad* de los lenguajes es un conjunto de lenguajes. Por ejemplo, “es de contexto libre” es el conjunto de lenguajes generados por CFG. “Es vacío” es el conjunto  $\{\emptyset\}$ .

**Definición 8.7.** Una propiedad se llama *no trivial* si hay lenguajes con la propiedad y lenguajes sin la propiedad.

**Teorema 8.11** (Rice). Toda propiedad no trivial de los lenguajes R.E. es no decidible.

Un ejemplo del teorema de Rice sería detectar si un lenguaje es regular, de contexto libre, etc. es no-decidible. Que es lo mismo que si  $P = \{M : \mathcal{L}(M) \text{ tiene la propiedad } P\}$ , entonces  $L$  no es recursivo.

La estrategia para demostrar el teorema de Rice es la siguiente:

- Supondremos que nos dan el lenguaje R.E. como una TM, decidir la propiedad es determinar dado  $M$  si  $\mathcal{L}(M)$  tiene la propiedad ( $M \in L_p$ )
- Reducimos  $L_u \leq L_p$ ; como  $L_u$  no es decidible, no es decidible  $L_p$ .

Es importante destacar, que el teorema de Rice no nos dice nada acerca de la máquina de Turing. **Es sólo para lenguajes, no para TM**

*Demostración.* Supongamos primeramente  $\emptyset \notin P$  (el lenguaje vacío no tiene la propiedad  $P$ ). Reducimos  $L_u$  a  $L_P$  mediante la siguiente construcción: “Sea  $L$  un lenguaje,  $L \in P$ , y  $L = \mathcal{L}(M_L)$  para una TM  $M_L$ . Notar que  $M_L$  es fijo”. Dada una instancia  $\langle M, \omega \rangle$  de  $U$ , construyo  $M'$ :

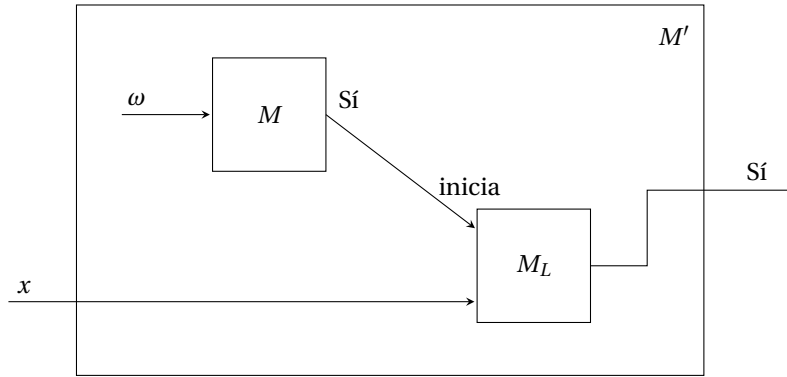


Figura 8.10: La máquina de Turing  $M'$  emula una máquina de Turing universal, que es no decidable. Note que si  $M$  acepta, echa a andar  $M_L$  sobre  $M'$ .

Notamos que:

$$\left. \begin{array}{l} \text{Si } \omega \in \mathcal{L}(M) \Rightarrow \mathcal{L}(M') = L \\ \text{Si } \omega \notin \mathcal{L}(M) \Rightarrow \mathcal{L}(M') = \emptyset \end{array} \right\} \begin{array}{l} \text{Si pudiera decidir si } \mathcal{L}(M) \in P, \text{ podría decidir si } \omega \in \mathcal{L}(M), \\ \text{pero } L_u \text{ no es decidable} \end{array}$$

Falta el caso  $\emptyset \in P$ . En tal caso, como  $P$  no es trivial,  $\emptyset \notin \bar{P}$ , y  $\bar{P}$  es una propiedad no trivial que no incluye  $\emptyset$ . Por el caso anterior,  $L_{\bar{P}}$  no es decidable, pero  $L_{\bar{P}} = \bar{L}_P$ . Como  $\bar{L}_P$  no es decidable,  $L_P$  no es decidable (haciendo este truco, podríamos decidir  $L_u$ , pero  $L_u$  no es decidable, por lo tanto, se origina una contradicción).  $\square$

Ojo, habla del lenguaje aceptado por  $M$ , no de  $M$ .

## 8.5. Problema de Correspondencia de Post (PCP)

Una instancia<sup>2</sup> de PCP es una colección de pares de palabras  $(\alpha_i, \beta_i)$  sobre  $\Sigma$ , siendo la pregunta si hay una secuencia finita  $\langle i_1, i_2, \dots, i_n \rangle$  tal que:

$$\alpha_{i_1} \alpha_{i_2} \cdots \alpha_{i_n} = \beta_{i_1} \beta_{i_2} \cdots \beta_{i_n} \quad (8.5.1)$$

**Ejemplo 8.1.** [2] Sea  $\Sigma = \{0, 1\}$ . Considere el Cuadro 8.3, que define todos los  $\alpha_i$ 's y  $\beta_i$ 's de nuestro PCP. Para la secuencia finita  $\langle i_1, i_2, i_3, i_4 \rangle = \langle 2, 1, 1, 3 \rangle$  vemos que este PCP si tiene solución. Esto lo podemos verificar concatenando los strings correspondientes en orden para ambas lsitas:

$$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3 = 101111110 \quad (8.5.2)$$

Note que esta solución no es única, ya que  $\langle i_1, i_2, \dots, i_8 \rangle = \langle 2, 1, 1, 3, 2, 1, 1, 3 \rangle$  es otra solución.  $\square$

Resulta que PCP no es decidable. Demostración es vía  $L_u \leq \text{PCP}$ . Idea general es que al ir construyendo la secuencia  $\alpha_{i_1} \alpha_{i_2} \cdots$ , vamos construyendo IDs de  $M$ , y en  $\beta_{i_1} \beta_{i_2} \cdots$ , vamos construyendo el ID siguiente de  $M$ , simulando una movida. Para que los  $\alpha$ 's “alcancen” las  $\beta$ 's, tienen que copiar el ID actual y los  $\beta$  generar el siguiente, ... Si en los  $\beta$ 's acepta, damos la posibilidad de terminar.

Podemos usar esto para demostrar cosas más terrenales...

<sup>2</sup>Una instancia del problema son sus datos de entrada.

$i$	$\alpha_i$	$\beta_i$
1	1	111
2	10111	10
3	10	0

Cuadro 8.3: Instancia PCP del Ejemplo 8.1. Corresponde a las definiciones de los  $\alpha_i$ 's y  $\beta_i$ 's

**Teorema 8.12.** No es decidible si una CFG es ambigua.

**Demostración.** Reducimos  $\text{PCP} \leq \text{CFG ambigua}$ . Sea  $(\alpha_i, \beta_i)$  una instancia de PCP. La gramática:

$$S \rightarrow A \mid B, \quad A \rightarrow iA\alpha_i \mid \epsilon, \quad B \rightarrow iB\beta_i \mid \epsilon \quad (8.5.3)$$

es ambigua si y solo si esa instancia de PCP tiene solución.  $\square$

## Capítulo 9

# Problemas Intratables

### 9.1. P vs NP

---

Las funciones polinomiales son cerradas respecto composición.

Si  $p(n)$  y  $q(n)$  son polinomios,  $p \circ q(n) = p(q(n))$  también es un polinomio.

Si el tiempo de ejecución de un algoritmo no está acotado por un polinomio en el tamaño de los datos, no hay esperanza de resolver instancias más “grandes”.

#### 9.1.1. Máquinas de Turing Deterministas vs otros modelos

Para cada modelo de computación podemos definir una medida “razonable” de “tiempos de ejecución”.

- Pasos de una TM determinista.
- Número de instrucciones ejecutadas por RAM.
- Número de llamadas de función en cálculo  $\lambda$ .
- ⋮

Con un poco de cuidado, si el modelo  $A$  toma tiempo  $T$ , podemos simularlo en tiempo  $p(T)$  en el modelo  $B$ , donde  $p()$  es un polinomio (por ejemplo, [simular el funcionamiento de una RAM en una TM](#)). Misterio: En todos los casos, el grado de  $p$  es más bien chico...

**Importante:** Todo algoritmo que tenga solución en un tiempo polinomial  $p$ , podrá ser simulado por cualquier otro modelo de computación que tenga solución en tiempo polinomial  $q$ .

Como podemos “traducir” entre modelos de computación pagando un costo en “tiempo” acotado por un polinomio  $\rightsquigarrow$  podemos obviar el modelo exacto (no hablamos de tiempo cronológico, sino polinomial).

Problemas que no admiten solución en “tiempo” acotado por un polinomio en el tamaño de la entrada claramente no son solubles en la práctica.

Estudio de esta problemática es la *teoría de complejidad*. Se habla de la *complejidad* de un algoritmo como la función (que acota) el uso de recursos en función del tamaño del problema. Recursos = tiempo, para nosotros (en general, es el recurso más importante). Decimos que un **problema tiene solución eficiente** si el tiempo está acotado por un polinomio.

[Algunas Observaciones hasta el momento:](#)

- La complejidad del algoritmo es la cota (en tiempo polinomial) en que el algoritmo logra resolver el problema.
- Complejidad del algoritmo es cuanto se demora en función de la cantidad de datos de entrada.

Para una TM no determinista, consideramos el tiempo de ejecución como el número de pasos de la computación más corta (el camino más corto, y siempre elige el camino más corto) que lleva a aceptar.

**Definición 9.1 (P).** Un problema está en P (determinista polinomial) si hay una TM determinista que determina si  $\sigma$  pertenece al lenguaje en un número de pasos acotado por un polinomio en  $|\sigma|$ . En otras palabras, se dice que un problema está en P si hay un algoritmo que lo resuelve en tiempo polinomial.

**Definición 9.2 (NP).** Un problema está en NP (no determinista polinomial) si hay una TM no determinista que acepta  $\sigma$  en un número de pasos acotado por un polinomio en  $|\sigma|$ .

Notar que las definiciones 9.1 y 9.2 pueden depender de la “codificación del problema.”

Es claro que  $P \subseteq NP$  (todo problema en P está en NP) ¿Es  $P = NP$ ? Nadie sabe...

Descripción alternativa: “Adivinar” (en tiempo polinomial) una solución, luego verificar que es solución en tiempo determinista polinomial. Es claro que esto es  $\subseteq NP$ , hay que demostrar que son iguales  $\rightsquigarrow$  La “solución” a adivinar es el certificado<sup>1</sup> (de que  $\sigma \in L_P$ ), se puede verificar en tiempo polinomial.

Ejemplos de problemas en P y NP:

- NP: Resolver un sudoku, adivinando dónde colocar todos los números.
- P: Encontrar algún algoritmo que calcule en tiempo polinomial dónde colocar los números del sudoku.

### 9.1.2. Reducción de problemas en tiempo polinomial

**Definición 9.3.** Un problema tiene solución eficiente si hay un algoritmo que decide si  $\sigma \in L_P$  en tiempo acotado por un polinomio en  $|\sigma|$ . (En forma poco precisa se dice que si  $f(n)$  no está acotada por un polinomio, es exponencial)

**Definición 9.4.** Se dice que P se reduce polinomialmente a Q ( $P \leq_p Q$ ) si hay un algoritmo que traduce una instancia  $\sigma$  de P en una instancia de Q con tiempo polinomial  $p$ . Las reducciones anteriores nos decían: “podemos hacer la reducción, pero el tiempo no importa”. En este caso, si.

Notar que si  $P \leq_p Q$ , y Q tiene solución eficiente, P tiene solución eficiente (funciona de la misma forma que vimos en veces anteriores. Véase la Figura 9.1). Además, si  $A \leq_p B$  y  $B \in NP$ , entonces  $A \in NP$ . De la misma forma, si  $A \leq_p B$ , y  $B \in P$ , entonces  $A \in P$ .

### 9.1.3. Definición de un problema NP-completo

**Definición 9.5.** Un problema se llama NP-duro (NP-hard) si todos los problemas en NP se reducen polinomialmente a él (H en la Figura 9.2).

**Definición 9.6.** Un problema se llama NP-completo (NP-complete) o completo para NP si es NP-duro y está en NP (Problema C en la Figura 9.3).

Notar que si se halla un problema NP-completo C con solución determinista polinomial, automáticamente  $P = NP$ .

<sup>1</sup> Certificado es la solución del problema.

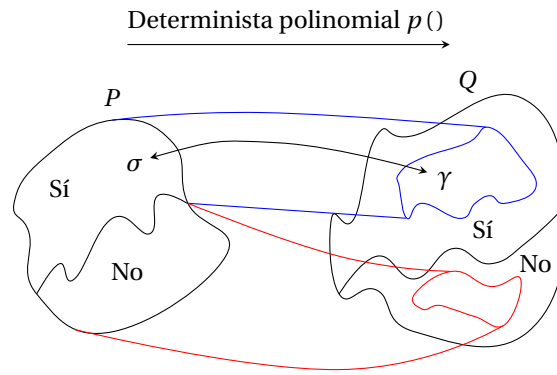


Figura 9.1: En el futuro, nos interesarán aplicar las reducciones que vimos anteriormente. Note que  $|\gamma| \leq p(|\sigma|)$

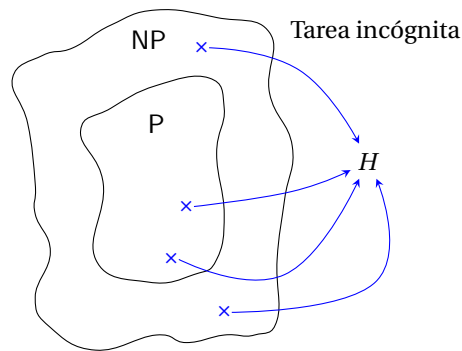


Figura 9.2: Representación gráfica de un problema NP-duro. Si podemos reducir todos los problemas en NP a un problema  $H$ , entonces el problema es NP-duro.

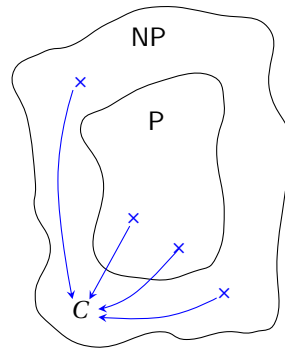


Figura 9.3: Representación gráfica de un problema NP-completo.

**Observación:** Si un problema  $H$  es NP-completo o NP-duro, es claro que no está en  $P$ , siempre que  $NP \neq P$ .

## 9.2. Algunos problemas NP-completos

**Teorema 9.1.** Si un problema NP-completo  $A$  está en  $P$ ,  $P = NP$ .

*Demostración.* Ejercicio verlo.

□

Notar que  $n^{\log_2(n)}$  no es polinomial ( $\log_2(n)$  crece al infinito, no está acotado por un polinomio)

$n^{\log_2(n)} = 2^{(\log_2(n))^2}$ , crece más lento que  $a^n$  para todo  $a > 1$  ya que  $a^n = 2^{n \log_2(a)}$ ,  $n \log_2(a)$  crece más rápido que  $(\log_a(n))^2$ .

### 9.2.1. Satisfiability problem (SAT)

Dada una fórmula lógica  $\phi(x_1, \dots, x_n)$  (escrita con variables  $\wedge, \vee, \neg$ ), determinar si hay una asignación de valores (verdadero o falso) a las variables  $x_i$  tal que  $\phi(x_1, \dots, x_n)$  sea verdadera (Satisfiability)

Plan:

- SAT es NP-completo (TM no determinista  $\leq_p$  SAT, teorema de Cook).
- 3SAT es NP-completo (SAT  $\leq_p$  3SAT)

⋮

**Definición 9.7.** Una variable o la negación de ella se llama literal, se anota  $x$  para la variable y  $\bar{x}$  para  $\neg x$ . Una cláusula es una disyunción<sup>2</sup> de literales, como  $x \vee \bar{y} \vee z$ .

**Definición 9.8.** Una fórmula lógica está en forma normal conjuntiva (CNF<sup>3</sup>) si es la conjunción<sup>4</sup> de cláusulas.

- CSAT: La fórmula lógica  $\phi(x_1, \dots, x_n)$  en CNF es satisfacible. CSAT es NP-completo (SAT  $\leq_p$  CSAT)
- $k$ -SAT: La fórmula lógica  $\phi(x_1, \dots, x_k)$  en CNF, donde cada cláusula tiene  $k$  literales.
- 3SAT es NP-completo.
- 1SAT, 2SAT tienen solución lineal (!).

**Teorema 9.2 (Cook).** SAT es NP-completo.

*Demostración.* SAT  $\in$  NP.

- SAT  $\in$  NP: Adivinar valores para las variables, calcular el valor de la expresión.
- TM no determinista  $M, \omega, p(n)$ ; determinista si  $\omega \in \mathcal{L}(M) \leq_p$  SAT con  $M$  aceptando en a lo más  $p(|\omega|)$  pasos.

Idea general: Armar una matriz de expresiones:

$$\begin{array}{cccc}
 x_{01} & x_{02} & \cdots & x_{0p(|\omega|)} \\
 x_{11} & x_{12} & \cdots & x_{1p(|\omega|)} \\
 \vdots & & & \\
 x_{p(|\omega|)1} & \cdots & \cdots & x_{p(|\omega|)p(|\omega|)}
 \end{array} \tag{9.2.1}$$

Observaciones:

- Cada fila de la matriz representa una descripción instantánea de la máquina de Turing.
- La primera fila es especial: ID  $M$  apronta a procesar  $\omega$
- Para la fila  $i$  a  $i + 1$ : movida legal de  $M$  a  $M$  acepté;  $ID_{i+1} = ID_i$ .
- Revisar si el estado es de aceptar.

<sup>2</sup>Una disyunción es un OR

<sup>3</sup>no confundir con Chomsky Normal Form

<sup>4</sup>Una conjunción es un AND



Escribir esta expresión toma tiempo polinomial (en  $M, \omega$ )

□

**Teorema 9.3.**  $SAT \leq_p CSAT$

*Demostración.* Aplicar de Morgan para llevar negaciones a variables, luego distribuir para lograr CNF,

Lo enredado es hacerlo en tiempo polinomial...

□

**Teorema 9.4.**  $CSAT \leq_p 3SAT$

*Demostración.* Problemas son cláusulas con más de 3 literales y cláusulas con menos de 3 literales.

- Menos de 3 literales, por ejemplo:  $x \vee y \rightsquigarrow (x \vee y \vee x_1) \wedge (x \vee y \vee \bar{x}_1)$  (con nueva variable  $x_1$ )
- Más de tres literales, por ejemplo:  $u \vee v \vee w \vee x \rightsquigarrow (u \vee v \vee x_1) \wedge (\bar{x}_1 \vee w \vee x)$

□

### 9.2.2. Independent Set (IS)

**Definición 9.9.** Sea  $G = (V, E)$  un grafo. Un subconjunto  $I$  de los vértices es un conjunto independiente (independent set) si no hay arcos entre vértices en  $I$ .  $I$  es maximal si no hay conjunto independientes mayores en  $G$ .

Por ejemplo, considere el grafo de la Figura 9.4:

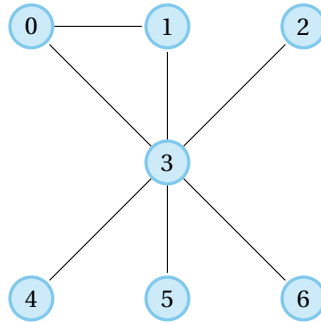


Figura 9.4: Grafo de avberrio

Entonces, de acuerdo a la definición 9.9, se tiene que el conjunto independiente es  $I = \{0, 2, 4, 5, 6\}$ , o bien,  $I = \{1, 2, 4, 5, 6\}$ . Las Figuras 9.5a y 9.5b muestran con más detalles (usando colores).

**Problema** Dado  $G = (V, E)$  y  $k \leq |V|$ , ¿tiene  $G$  un IS de tamaño  $\leq k$ ?

**Teorema 9.5.** IS es NP-completo.

*Demostración.* Recordemos que, para poder demostrar que un problema es NP-completo, tenemos que demostrar dos cosas: que está en NP, y que es NP-duro.

Para el primer caso, sólo tenemos que elegir  $k$  vértices, y comprobar que no existen conexiones entre ellos. Hacer esto toma una complejidad polinomial de grado 2. En consecuencia, este problema está en NP ( $P \subseteq NP$ ).

Para demostrar que el problema es NP-duro, hacemos la reducción en tiempo polinomial  $3SAT \leq_p IS$ . Consideremos el siguiente 3SAT como ejemplo:

$$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_4) \wedge (x_4 \vee x_1 \vee \bar{x}_2) \quad (9.2.2)$$

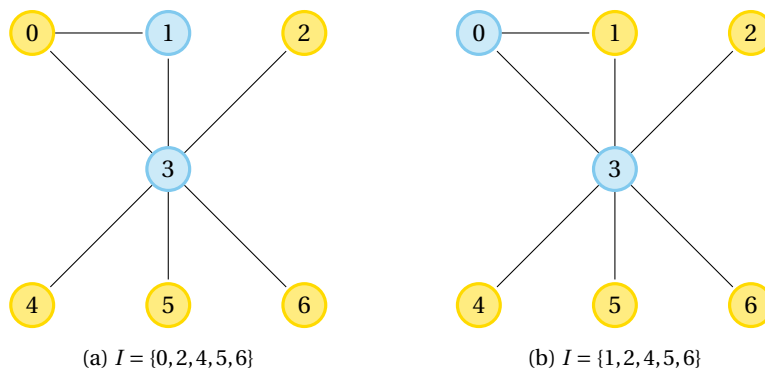


Figura 9.5: Los nodos amarillos corresponden a vértices que pertenecen a  $I$ .

**Idea:** Literales  $x$ ,  $y$  no son verdaderos a la vez:  $x$  conectado con  $y$ . Sólo un literal verdadero por cláusula  $\rightsquigarrow k$  es el número de cláusulas (Figura 9.6).

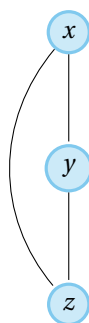


Figura 9.6: Para cada cláusula  $(x \vee y \vee z)$  tenemos que hacer este tipo de construcción.

Para cada variable, conectar  $x$  y  $\bar{x}$  (Figura 9.7)



Figura 9.7: Conectamos cada variable  $x$  con su negación  $\bar{x}$

Luego, aplicando los criterios anteriores sobre el ejemplo 3SAT de la ecuación (9.2.2), podemos formar el grafo de la Figura 9.8.

El grafo de la Figura 9.8 tiene un IS de tamaño 3 si y sólo si en la fórmula original podemos asignar el valor verdadero a un literal en cada cláusula, que si y sólo si la formula puede satisfacerse.

Finalmente, como IS es NP-duro y está en NP, se tiene que IS es NP-completo.  $\square$

**Importante** La reducción debe hacerse en tiempo polinomial. Si no podemos construir IS a partir de 3SAT en tiempo polinomial, fallamos.

### 9.2.3. Node Cover

Un “covering” de  $G = (V, E)$  considera todos los vértices/arcos. Un *edge covering* de  $G$  es un conjunto de arcos tal que todos los vértices de  $G$  están en un arco (Figura 9.9a). Un *node covering* es un

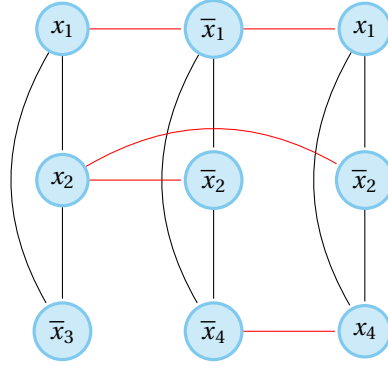


Figura 9.8: Construcción del 3SAT de la ecuación (9.2.2) usando los criterios mencionados previamente.

conjunto de vértices que aparecen en todos los arcos (Figura 9.9b).

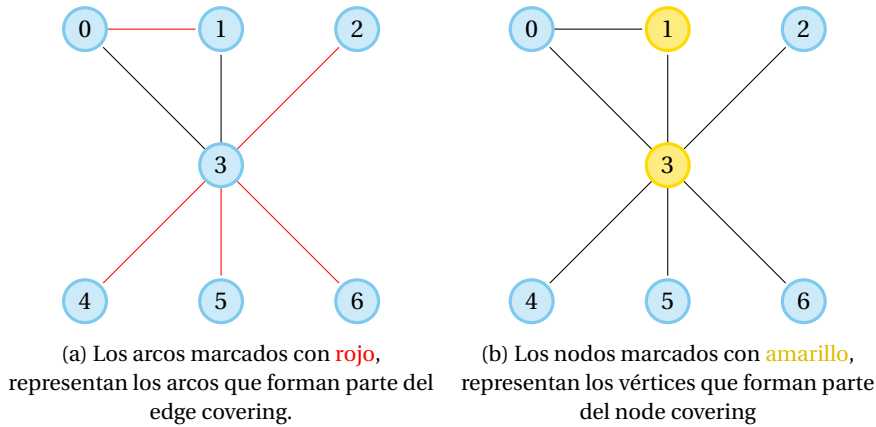


Figura 9.9: Nótese que tanto los arcos como los nodos que forman parte del edge covering y el node covering respectivamente, son sólo 1 ejemplo. En el grafo de la Figura 9.4 hay otros conjuntos.

**Problema Node Cover (NC):** Dado  $G = (V, E)$  y  $k \leq |V|$ , ¿Hay un node cover de  $G$  de tamaño  $k$ ?

**Teorema 9.6.** NC es NP-completo.

*Demostración.* Como toda demostración de problema NP-completo, tenemos que demostrar que nuestro problema es NP y además, es NP-duro.

Como primer paso, demostrar que NC pertenece a NP, sólo basta con comprobar que todos los nodos cumplan con la definición de node covering (trivial). Esto nos dice que  $NC \in P$ .

El complemento de un conjunto independiente (IS) es un NC. O sea:

$$IS \leq_p NC \quad (9.2.3)$$

□

#### 9.2.4. ¿Por qué hacemos estas reducciones?

Lo que logramos hasta el momento es pasar de demostraciones con TM a otras más abstractas:

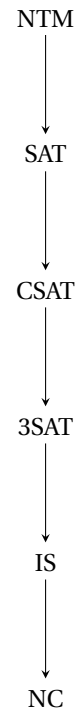


Figura 9.10: La gracia del problema SAT, es que nos permitió pasar de hacer demostraciones de problemas NP-completos con máquinas de Turing a demostraciones más abstractas.

# Bibliografía

- [1] Introduction to Automata Theory, Languages and Computation Second Edition, by John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman – Sección 6.3, Figura 6.8.
- [2] Introduction to Automata Theory, Languages and Computation Second Edition, by John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman – Ejemplo 9.13