Introducción 3era Parte

Introducción a la Lógica y la Computación (3era Parte)

Docentes: Badano, Bustos, Costamagna, Tellechea, Zigaran

Año 2024

Que vamos a estudiar en esta 3era parte del curso?

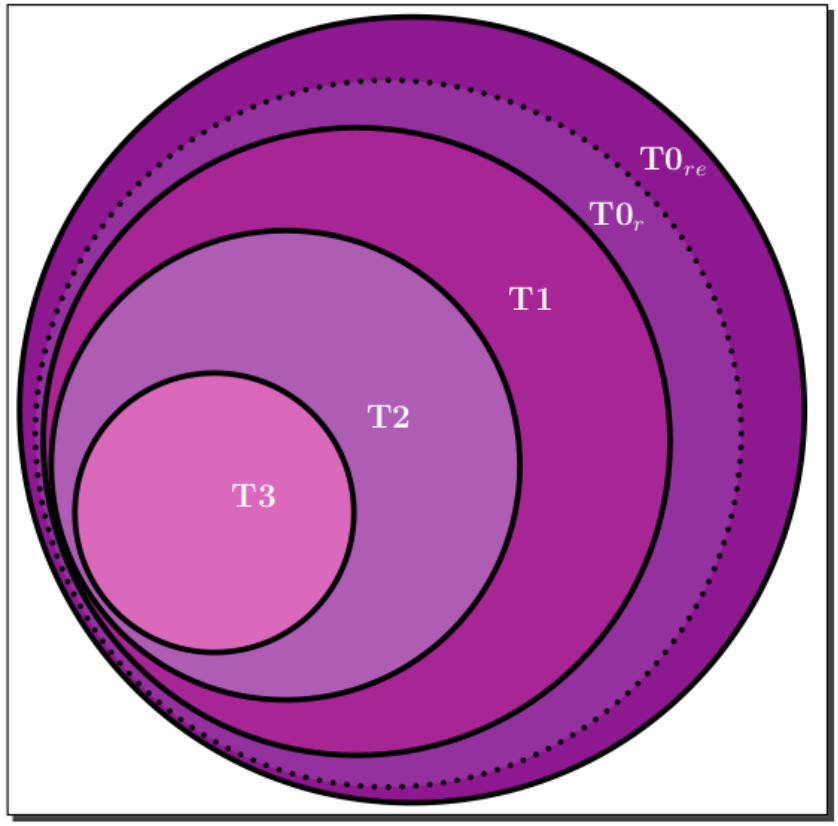
- Vamos a estudiar la noción de **lenguaje formal** y clasificarlos según qué tan dificiles son de computar, en el sentido de qué tan complejo es el modelo computacional necesario para computarlos.

- Veremos que existen lenguajes más complejos de computar que otros. Por ende, aquellos que tengan la misma dificultad son clasificados a una misma **clase de lenguajes**.

- En particular, estudiaremos exclusivamente la primera clase de lenguajes propuesta en la "**Jerarquia de Chomsky**".

Jerarquia de Chomsky

- Lenguajes Regulares (T3)
- Lenguajes Independientes de Contexto (T2)
- Lenguajes Sensibles de Contexto (T1)
- Lenguajes Recursivamente Enumerables (T0)



Modelos Computacionales

Existen dos tipos:

- Modelos Generativos $\rightarrow$ **Gramáticas**
- Modelos Reconocedores $\rightarrow$ **Autómatas**

Cada clase de lenguajes en la Jerarquía de Chomsky está caracterizada por el modelo computacional (más simple) necesario para computarla:

- **Lenguajes Regulares:** Autómatas Finitos y Gramáticas Regulares.
- **Lenguajes Independientes de Contexto:** Autómatas Finitos con 1-Pila y Gramáticas Independiente de Contexto.
- **Lenguajes Sensibles de Contexto:** Autómatas Finitos con n-Pilas y Gramáticas Sensibles de Contexto.
- **Lenguajes Recursivamente Enumerables:** Máquina de Turing y Gramáticas sin restricciones.

Modelos Computacionales

- En este curso estudiaremos solamente la clase de los **"Lenguajes Regulares"** y sus dos modelos computacionales:
  - **Automatas Finitos**
  - **Gramáticas Regulares**
- En 4to año estudiarán en detalle la siguiente clase de lenguajes **"Lenguajes Independientes de Contexto"** y sus dos respectivos modelos computacionales:
  - **Autómatas Finitos con 1-pila**
  - **Gramáticas Independiente de Contexto**

Alfabetos, Cadenas, Lenguajes y sus Operadores

Introducción a la Logica y la Computación (3era Parte)

Docentes: Badano, Bustos, Costamagna, Tellechea, Zigaran

Año 2024

Alfabetos y Cadenas

- Un **alfabeto** es cualquier conjunto finito de símbolos arbitrarios y se suelen denotar con la letra griega Σ. Por ejemplo: $\Sigma_1 = \{a, b, c\}$ es un alfabeto con tres símbolos, mientras que $\Sigma_2 = \{\&, \#\}$ es un alfabeto con dos símbolos.
- Una **cadena** del alfabeto Σ es cualquier secuencia finita de símbolos de Σ. Las denotaremos con letra griegas α, β, γ. Por ejemplo: $\alpha =$ "*abb*" es un cadena del alfabeto Σ_1, mientras que $\beta =$ "&\#\#&" es un cadena del alfabeto Σ_2.
- Por último, denotaremos con Σ^* al conjunto de todas las cadenas posibles del alfabeto Σ (el universo de cadenas del alfabeto). Y con ϵ denotaremos a la única cadena que no contiene símbolos, llamada cadena vacia.

Operadores de Cadenas

Definimos cuatro operadores de cadenas:

- Concatenación
- Potencia
- Longitud
- Reversa

Concatenación de Cadenas

Sea $\alpha, \beta \in \Sigma^*$, definimos la concatenación:

$$\alpha.\beta = \begin{cases} \alpha & \text{si } \beta = \epsilon \\ \beta & \text{si } \alpha = \epsilon \\ a_1 \dots a_n b_1 \dots b_m & \text{si } \alpha = a_1 \dots a_n \text{ y } \beta = b_1 \dots b_m \end{cases}$$

Por ejemplo, si $\alpha =$ "*aa*" y $\beta =$ "*bb*", entonces $\alpha.\beta =$ "*aabb*".

Algunas propiedades de la operación:

1. Asociativa: $\alpha.(\beta.\gamma) = (\alpha.\beta).\gamma$
2. No conmuta, pues en gral $\alpha.\beta \neq \beta.\alpha$
3. El elemento neutro de la operación es la cadena vacia, pues $\alpha.\epsilon = \epsilon.\alpha = \alpha$

De aquí en adelante, denotamos la concatenacion de α y β, escribiendo simplemente $\alpha\beta$ (omitimos el ".")

Potencia de una Cadena

Sea $\alpha \in \Sigma^*$ y $n \in \mathbb{N}$, definimos la potencia n-esima de la cadena α:

$$\alpha^n = \begin{cases} \epsilon & \text{si } n = 0 \\ \underbrace{\alpha \ldots \alpha}_{\text{n-veces}} & \text{si } n \geq 1 \end{cases}$$

Por ejemplo: $\alpha = ab$, entonces $\alpha^3 = ababab$.

Algunas propiedades de la operación:

1. $\alpha^n \alpha^m = \alpha^{n+m}$
2. $(\alpha^n)^m = \alpha^{n.m}$

Longitud de una Cadena

Sea $\alpha \in \Sigma^*$, definimos la longitud de la cadena α:

$$|\alpha| = \begin{cases} 0 & \text{si } \alpha = \epsilon \\ n & \text{si } \alpha = a_1 \dots a_n \end{cases}$$

Por ejemplo: $|abbab| = 5$ y $|\epsilon| = 0$.

Algunas propiedades de la operación:

1. $|\alpha\beta| = |\alpha| + |\beta|$
2. $|\alpha^n| = n.|\alpha|$
3. $|\alpha^{n+m}| = |\alpha^n| + |\alpha^m|$
4. $|(\alpha^n)^m| = |\alpha|^{n.m}$

Finalmente, con $|\alpha|_a$ denotaremos la cantidad de ocurrencias del símbolo "a" en la cadena α. Por ejemplo: $|abbab|_a = 2$ y $|abbab|_b = 3$.

Reversa de una Cadena

Sea $\alpha \in \Sigma^*$, definimos la reversa de la cadena α:

$$\alpha^R = \begin{cases} \epsilon & \text{si } \alpha = \epsilon \\ a_n \dots a_1 & \text{si } \alpha = a_1 \dots a_n \end{cases}$$

Por ejemplo: si $\alpha = abb$, entonces $\alpha^R = bba$.

Algunas propiedades de la operación:

1. $(\alpha\beta)^R = \beta^R\alpha^R$
2. $(\alpha^R)^R = \alpha$
3. $|\alpha^R| = |\alpha|$

Subcadena, Prefijo y Sufijo de una cadena

Sean $\alpha, \alpha' \in \Sigma^*$, diremos que:

- α' es **subcadena** de α sii $\exists \beta_1, \beta_2 \in \Sigma^*$ tal que $\alpha = \beta_1 \alpha' \beta_2$.
- α' es **prefijo** de α sii $\exists \beta \in \Sigma^*$ tal que $\alpha = \alpha' \beta$.
- α' es **sufijo** de α sii $\exists \beta \in \Sigma^*$ tal que $\alpha = \beta \alpha'$.

Por ejemplo: si $\alpha =$ "*abb*", entonces "*b*" es subcadena, "*ab*" prefijo y "*bb*" sufijo de α.

Lenguajes

- Un **lenguaje** en el alfabeto Σ es cualquier conjunto de cadenas de Σ. Más formalmente, un **lenguaje** es cualquier conjunto $L \subseteq \Sigma^*$.
- Por ejemplo, si $\Sigma = \{a, b\}$:
  - $L_1 = \emptyset$ es el lenguaje que no contiene cadenas (lenguaje vacio).
  - $L_2 = \Sigma^*$ es el lenguaje de todas la cadenas en el alfabeto.
  - $L_3 = \{aa, ab, ba, bb\}$ es el lenguaje de todas la cadenas de longitud 2.
  - $L_4 = \{\epsilon, a, a^2, a^3, a^4, \dots\}$ es el lenguaje de todas la cadenas que no tienen un símbolo "b".
  - $L_5 = \{\alpha \in \Sigma^* : |\alpha|_a \textit{ es par}\}$ es el lenguaje de todas la cadenas que tienen una cantidad par de símbolos "a".
  - $L_6 = \{\alpha \in \Sigma^* : \alpha = \alpha^R\}$ es el lenguaje de todas la cadenas capicuas.
- Notar que un lenguaje puede ser finito (como L_1 y L_3) o infinito (como L_2, L_4, L_5, L_6).

Operadores de Lenguajes

Como los lenguajes son conjuntos (de cadenas), entonces todo operador de conjuntos es tambien un operador de lenguajes.

- Si L_1 y L_2 son lenguajes, entonces:
  - $L_1 \cup L_2$ es un lenguaje (unión).
  - $L_1 \cap L_2$ es un lenguaje (intersección).
  - $L_1 - L_2$ es un lenguaje (resta).
  - $\overline{L_1}$ es un lenguaje (complemento).

Además, definimos los operadores de concatenación, potencia, clausura y reversa de lenguajes.

Concatenación de Lenguajes (L_1L_2)

Sean $L_1, L_2 \subseteq \Sigma^*$, definimos la **concatenación**:

$$L_1.L_2 = \{\alpha\beta : \alpha \in L_1 \ y \ \beta \in L_2\}$$

Por ejemplo, si $L_1 = \{a, aa\}$ y $L_2 = \{b, bb\}$, entonces $L_1.L_2 = \{ab, abb, aab, aabb\}$.

También omitiremos escribir "." para denotar la operación de concatenación de lenguajes.

Algunas propiedades de la concatenación:

1. Elemento absorvente: $L\emptyset = \emptyset L = \emptyset$
2. Elemento neutro: $L\{\epsilon\} = \{\epsilon\}L = L$
3. Asociatividad: $L_1(L_2L_3) = (L_1L_2)L_3$
4. Distributividad con $\cup$ por izquierda y por derecha: $L_1(L_2 \cup L_3) = (L_1L_2) \cup (L_1L_3)$ y $(L_2 \cup L_3)L_1 = (L_2L_1) \cup (L_3L_1)$
5. No es conmutativa y tampoco es distributiva con $\cap$

Potencia de Lenguajes (L^n)

Sea $L \subseteq \Sigma^*$ y $n \in \mathbb{N}$, definimos la **potencia n-esima** de L:

$$L^n = \begin{cases} \{\epsilon\} & \text{si } n = 0 \\ \underbrace{L \ldots L}_{\text{n-veces}} & \text{si } n \geq 1 \end{cases}$$

L^n es el conjunto de todas las posibles concatenaciones de n cadenas de L.

Por ejemplo, si $L = \{a, bb\}$, entonces $L^2 = \{aa, abb, bba, bbbb\}$.

Clausura de un Lenguaje (L^* y L^+)

Sean $L \subseteq \Sigma^*$, definimos la **clausura** de L:

$$L^* = \{\alpha_1 \dots \alpha_n : \alpha_i \in L \ y \ n \geq 0\}$$

L^* es el conjunto de todas las posibles concatenaciones de cadenas de L. Por lo tanto, L^* es un lenguaje infinito (salvo que $L = \emptyset$ ó $L = \{\epsilon\}$) y siempre contiene a la cadena ϵ.

Por ejemplo, si $L = \{a\}$, entonces $L^* = \{\epsilon, a, aa, aaa, aaaa, \dots\}$

Tambien, definimos la **clausura positiva** de L:

$$L^+ = \{\alpha_1 \dots \alpha_n : \alpha_i \in L \ y \ n \geq 1\}$$

Siguiendo el ejemplo anterior, $L^+ = \{a, aa, aaa, aaaa, \dots\}$

Notar que $L^+ = L^* - \{\epsilon\}$, para todo L

Propiedades de la Clausura

Sea $L \subseteq \Sigma^*$, entonces se cumplen:

1. $L^* = \bigcup_{i=0}^{\infty} L^i$
2. $L^+ = \bigcup_{i=1}^{\infty} L^i$
3. $L^* L^* = L^*$
4. $L^+ L^+ = L^+$
5. $(L^*)^* = L^*$
6. $(L^+)^+ = L^+$
7. $L^+ = LL^*$
8. $(L^+)^* = L^*$
9. $(L^+)^* = L^*$

Reversa de un Lenguaje (L^R)

Sea $L \subseteq \Sigma^*$, definimos la **reversa** de L:

$$L^R = \{\alpha^R : \alpha \in L\}$$

L^R es el conjunto de todas las reversas de las cadenas de L.

Por ejemplo, si $L = \{a, ab, babb\}$ entonces $L^R = \{a, ba, bbab\}$

Algunas propiedades de la reversa:

1. $(L_1 \cup L_2)^R = {L_1}^R \cup {L_2}^R$
2. $(L_1 \cap L_2)^R = {L_1}^R \cap {L_2}^R$
3. $(L_1 L_2)^R = {L_2}^R {L_1}^R$
4. $(L^R)^R = L$
5. $(L^*)^R = (L^R)^*$
6. $(L^+)^R = (L^+)^*$

Precedencia de los Operadores

En asusencia de paréntesis, las operadores siguen el siguiente orden de precedencia:

1. Potencia, Reversa, Clausura
2. Concatención
3. Unión, Intersección, Resta

Resumen

- Definimos formalmente **la noción de lenguaje** a partir del concepto de alfabeto y cadenas del alfabeto.
- Definimos **operadores** a nivel de las cadenas y luego la extendimos a nivel de los lenguajes.
- Enunciamos las **principales propiedades** que satisfacen dichas operaciones y su **orden de precedencia**.

Lenguajes Regulares y Expresiones Regulares

Introducción a la Logica y la Computación (3era Parte)

Docentes: Badano, Bustos, Costamagna, Tellechea, Zigaran

Año 2024

Lenguajes Regulares

- Los **lenguajes regulares** son los lenguajes más simples dentro de la Jeraquia de Chomsky.
- Los **lenguajes regulares** son aquellos que se pueden construir (recursivamente) a partir de ciertos **lenguajes muy básicos o atómicos**, aplicando una **cantidad finita** de ciertos operadores de lenguajes.
- Más concretamente, dichas operadores son tres: **unión, concatenación y clausura**. Es por esto, que estos operadores son conocidos como los "operadores regulares", ya que, son los que permiten construir a todos los lenguajes regulares.

Definición Recursiva de Lenguajes Regulares

Sea Σ un alfabeto, entonces:

Casos Base:

- $\emptyset$ es un lenguaje regular
- $\{\epsilon\}$ es un lenguaje regular
- $\{a\}$ es un lenguaje regular, con $a \in \Sigma$

Casos Recursivos:
Si L_1 y $L2$ son lenguajes regulares, entonces:

- $L_1 \cup L_2$ es un lenguaje regular
- $L_1 L_2$ es un lenguaje regular
- ${L_1}^*$ es un lenguaje regular

Definición Recursiva de Lenguajes Regulares

Más formalmente, podemos definir como LR_k^Σ al conjunto de los lenguajes regulares que se obtienen a partir de los casos base aplicando hasta k operadores regulares:

$$LR_0^\Sigma = \{\emptyset\} \cup \{\{\epsilon\}\} \cup \{\{a\} : a \in \Sigma\}$$

$$\begin{aligned} LR_{k+1}^\Sigma = LR_k^\Sigma &\cup \{L_1 \cup L_2 : L_1, L_2 \in LR_k^\Sigma\} \\ &\cup \{L_1 L_2 : L_1, L_2 \in LR_k^\Sigma\} \\ &\cup \{L^* : L \in LR_k^\Sigma\} \end{aligned}$$

Para luego, definir como LR^Σ al conjunto de todos los lenguajes regulares:

$$LR^\Sigma = \bigcup_{k=0}^{\infty} LR_k^\Sigma$$

Ejemplos de Lenguajes Regulares

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

$L_1 = \{b^n aab^m : n, m \in \mathbb{N}\} \in LR^{\Sigma}$, pues

$$\begin{aligned} L_1 &= \{b^n aab^m : n, m \in \mathbb{N}\} \\ &= \{b^n : n \in \mathbb{N}\}\{aa\}\{b^n : m \in \mathbb{N}\} \\ &= \{b\}^*\{a\}\{a\}\{b\}^* \end{aligned}$$

$L_2 = \{b\alpha : \alpha \in \Sigma^*\} \in LR^{\Sigma}$, pues

$$\begin{aligned} L_2 &= \{b\}\{\alpha : \alpha \in \Sigma^*\} \\ &= \{b\}\Sigma^* \\ &= \{b\}\{a, b\}^* \\ &= \{b\}(\{a\} \cup \{b\})^* \end{aligned}$$

$L_3 = \{a^n b^m : n, m \in \mathbb{N}\} \in LR^{\Sigma}$, pues

$$\begin{aligned} L_3 &= \{a^n : n \in \mathbb{N}\}\{b^m : m \in \mathbb{N}\} \\ &= \{a\}^*\{b\}^* \end{aligned}$$

Finitud y Regularidad

Todo lenguaje finito es un lenguaje regular.

Teorema

Si L es finito, entonces $L \in LR^{\Sigma}$.

Demo. Si L finito entonces $L = \{\alpha_1, \ldots, \alpha_k\}$ con $\alpha_i \in \Sigma^*$ y $k \geq 0$.
Si $k = 0$, entonces $L = \emptyset \therefore L$ regular por caso base.
Si $k \geq 1$, entonces $L = \{\alpha_1\} \cup \cdots \cup \{\alpha_k\} \therefore L$ regular si $\{\alpha_i\}$ es regular.

Veamos que $\{\alpha_i\} \in LR^{\Sigma}$.

Si $\alpha_i = \epsilon \therefore \{\epsilon\} \in LR^{\Sigma}$ por caso base.
Si $\alpha_i = a_1 \ldots a_r$ con $r \geq 1$, entonces
$\{\alpha_i\} = \{a_1\} \ldots \{a_r\}$, luego
$\{a_j\} \in LR^{\Sigma}$ por caso base
$\therefore \{\alpha_i\} \in LR^{\Sigma}$.

Propiedades de los Lenguajes Regulares

Los lenguajes regulares son cerrados para los operadores de conjunto y la reversa.

Teorema

Si $L_1, L_2 \in LR^{\Sigma}$, *entonces:*

1. $L_1 \cup L_2 \in LR^{\Sigma}$ *(trivial por definición de lenguaje regular)*
2. $L_1 \cap L_2 \in LR^{\Sigma}$
3. $L_1 - L_2 \in LR^{\Sigma}$
4. $\overline{L_1} \in LR^{\Sigma}$
5. ${L_1}^R \in LR^{\Sigma}$

Demo. Las demostraciones las haremos más tarde en el práctico, ya que, para probarlo de forma relativamente sencilla, necesitamos un par de resultados teóricos aún no vistos hasta aquí.

Expresiones Regulares

- Las **expresiones regulares** son expresiones o fórmulas que sirven para describir lenguajes regulares (de allí su nombre) en forma muy legible y compacta.
- Con **expresiones regulares** podremos realizar aritmética de lenguajes regulares más fácilmente, ya que, en lugar de describir un conjunto de cadenas (regular), directamente podemos escribir la expresión regular que lo denota.
- Las **expresiones regulares** también se construyen recursivamente a partir de expresiones regulares básicas o atómicas y aplicando una cierta cantidad finita de operadores.

Definición Recursiva de Expresiones Regulares

Sea Σ un alfabeto, entonces:

Casos Base:

- $\emptyset$ es una expresión regular
- ϵ es una expresión regular
- a es una expresión regular, con $a \in \Sigma$

Casos Recursivos:
Si e_1 y e_2 son expresiones regulares, entonces:

- $e_1 + e_2$ es una expresión regular
- $e_1 e_2$ es un expresión regular
- ${e_1}^*$ es un expresión regular

Definición Recursiva de las Expresiones Regulares

Más formalmente, podemos definir como ER_k^Σ al conjunto de las expresiones regulares que se obtienen a partir de los casos base aplicando hasta k operadores:

$$ER_0^\Sigma = \Sigma \cup \{\emptyset, \epsilon\}$$

$$\begin{aligned} ER_{k+1}^\Sigma = ER_k^\Sigma &\cup \{e_1 + e_2 : e_1, e_2 \in ER_k^\Sigma\} \\ &\cup \{e_1 e_2 : e_1, e_2 \in ER_k^\Sigma\} \\ &\cup \{e^* : e \in ER_k^\Sigma\} \end{aligned}$$

Para luego, definir como ER^Σ al conjunto de todas las expresiones regulares:

$$ER^\Sigma = \bigcup_{k=0}^{\infty} ER_k^\Sigma$$

Lenguaje denotado por una Expresión Regular

Sea e una expresión regular, definimos como $L(e)$ al "lenguaje denotado por e" de la siguiente manera:

Casos Base:

- $L(\emptyset) = \emptyset$
- $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\}$ con $a \in \Sigma$

Casos Recursivos:

Si e_1 y e_2 son expresiones regulares, entonces:

- $L(e_1 + e_2) = L(e_1) \cup L(e_2)$
- $L(e_1 e_2) = L(e_1)L(e_2)$
- $L(e^*) = (L(e))^*$

Ejemplos de Expresión Regular y su Lenguaje Denotado

La expresión $e_1 = (a+b)^* a (a+b)^*$ es una expresión regular del alfabeto $\Sigma = \{a, b\}$.

Esta expresión denota al *"lenguaje de todas las cadenas que contienen al menos un símbolo a"*.

Verifiquemos esto último aplicando la definición recursiva del lenguaje denotado por una expresión regular:

$$\begin{aligned}
L((a+b)^* a (a+b)^*) &= L((a+b)^*) L(a) L((a+b)^*)) \\
&= (L(a+b))^* L(a) (L(a+b))^* \\
&= (L(a) \cup L(b))^* L(a) (L(a) \cup L(b))^* \\
&= (\{a\} \cup \{b\})^* \{a\} (\{a\} \cup \{b\}))^* \\
&= \{a, b\}^* \{a\} \{a, b\}^* \\
&= \{\alpha_1 a \alpha_2 : \alpha_1, \alpha_2 \in \Sigma^*\} \\
&= \{\alpha \in \Sigma^* : |\alpha|_a \geq 1\}
\end{aligned}$$

Equivalencia entre Lenguajes Regulares y Expresiones Regulares

Un lenguaje es regular si y solo si es denotado por una expresión regular.

Este resultado implica una ida y una vuelta:

1. Ida: Todo lenguaje regular tiene una expresión regular que lo denota.
2. Vuelta: Toda expresión regular denota un lenguaje regular.

Formalicemos técnicamente este resultado para su posterior prueba:

Teorema ($LR^{\Sigma} \equiv ER^{\Sigma}$)

1. *(Ida) Si* $L \in LR^{\Sigma}$, *entonces* $\exists e \in ER^{\Sigma}$ *tal que* $L(e) = L$.
2. *(Vuelta) Si* $e \in ER^{\Sigma}$, *entonces* $\exists L \in LR^{\Sigma}$ *tal que* $L(e) = L$.

Solo probaremos el inciso 1, pues 2 es análogo (ejercicio!).

Demo 1. Lo probamos por inducción en la forma del lenguaje regular L.

Equivalencia entre Lenguajes Regulares y Expresiones Regulares

Casos base:

- Si $L = \emptyset$, entonces tomamos $e = \emptyset$ y $L(\emptyset) = \emptyset$.
- Si $L = \{\epsilon\}$, entonces tomamos $e = \epsilon$ y $L(\epsilon) = \{\epsilon\}$.
- Si $L = \{a\}$, entonces tomamos $e = a$ y $L(a) = \{a\}$.

Casos Recursivos:

- Si $L = L_1 \cup L_2$, con $L_1, L_2 \in LR^{\Sigma}$, entonces por HI, $\exists e_1, e_2 \in ER^{\Sigma}$ tal que $L(e_1) = L_1$ y $L(e_2) = L_2$. Por lo tanto, podemos tomar $e = e_1 + e_2$, y luego, $L(e_1 + e_2) = L(e_1) \cup L(e_2) = L_1 \cup L_2$.
- Si $L = L_1 L_2$, con $L_1, L_2 \in LR^{\Sigma}$, entonces por HI, $\exists e_1, e_2 \in ER^{\Sigma}$ tal que $L(e_1) = L_1$ y $L(e_2) = L_2$. Por lo tanto, podemos tomar $e = e_1 e_2$, y luego, $L(e_1 e_2) = L(e_1)L(e_2) = L_1 L_2$.
- Si $L = {L_1}^*$, con $L_1 \in LR^{\Sigma}$, entonces por HI, $\exists e_1 \in ER^{\Sigma}$ tal que $L(e_1) = L_1$. Por lo tanto, podemos tomar $e = {e_1}^*$, y luego, $L({e_1}^*) = (L(e_1))^* = {L_1}^*$.

Resumen

- Dimos una definición recursiva de los lenguajes regulares (LR^{Σ}).
- Probamos que finitud implica regularidad.
- Dimos una definición recursiva de las expresiones regulares (ER^{Σ}) y su lenguaje denotado.
- Probamos que los lenguajes regulares y las expresiones regulares son equivalentes ($LR^{\Sigma} \equiv ER^{\Sigma}$).
- Entonces para probar que un lenguaje es regular, podemos dar directamente una expresión regular que lo denote.
- O si, simplemente queremos describir un lenguaje regular, en lugar de utilizar comprensión de conjuntos, podemos escribir directamente su expresión regular equivalente (notar que no es única! existen varias expresiones regulares distintas que denotan un mismo lenguaje).

Autómatas Finitos

Introducción a la Lógica y la Computación (3era Parte)

Docentes: Badano, Bustos, Costamagna, Tellechea, Zigaran

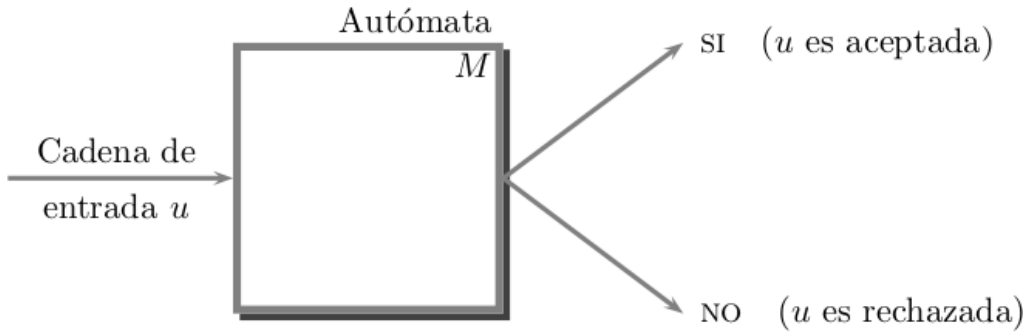
Año 2024

Autómatas Finitos (AF)

- Es un modelo computacional basado en la idea de **máquina automática secuencial** y cuyo poder computacional caracteriza a los lenguajes regulares.
- Primero, estudiaremos los autómatas finitos en si mismos, es decir, su funcionamiento, sus tipos y sus propiedades.
- Para luego, finalmente probar que es la máquina secuencial caracterizadora de los lenguajes regulares.

Qué computa un AF?

- Toma una cadena como input y luego de procesar todos sus símbolos devuelve si ésta es **"aceptada"** o **"rechazada"**.

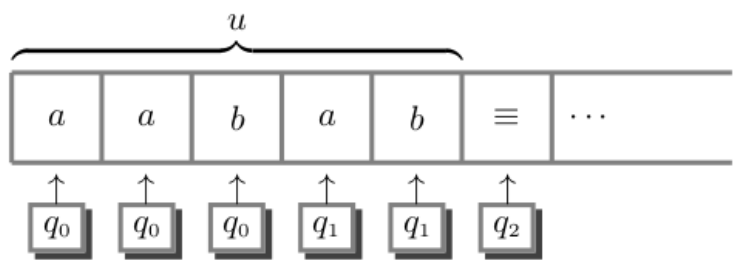


- Estudiaremos 3 tipos de autómatas finitos:
  - Determinísticos (**AFD**)
  - No-Determinísticos (**AFN**)
  - No-Determinísticos con transiciones espontáneas (**AFN**ϵ)
- Pero finalmente, todos **computacionalmente equivalentes** entre si.

Cómo funciona un AF?

- La máquina tiene un conjunto de **estados** posibles pero en todo momento se encuentra exactamente en uno de ellos.
- Tiene una cinta con celdas, llamada **cinta de entrada**, donde en cada celda se aloja un símbolo de la cadena a procesar, y un cabezal (de solo lectura) que apunta a la siguiente celda a procesar.
- Cada vez que el cabezal lee un símbolo de la cinta, la máquina ejecuta un cambio de estado, llamado **transición**. Así sucesivamente, hasta que el cabezal lee el último símbolo de la cadena a procesar.
- El estado al que arriba la máquina, luego de procesar toda la cadena, definirá si la cadena es **aceptada o rechazada**.

$u = aabab.$



Definición Autómata Finito Determinístico

Definición (AFD)

Un **Automata Finito Deterministico (AFD)** *es un 5-upla* $M = (\Sigma, Q, q_0, F, \delta)$ *donde:*

- Σ *es un alfabeto, llamado alfabeto de entrada.*
- Q *es un conjunto (no vacio y finito) de estados*
- $q_0 \in Q$, *llamado estado inicial*
- $F \subseteq Q$, *llamado conjunto de estados de aceptación o finales*
- $\delta : Q \times \Sigma \to Q$, *llamada función de transición, tal que,* $\delta(q, a) = q'$ *significa que si el autómata se encuentra en el estado "q" y el cabezal lee un símbolo "a" en la cinta de entrada, entonces el autómata transiciona al estado* q'.

Diremos que una cadena $\alpha \in \Sigma^*$ *es* **aceptada** *por el autómata* M, *si el estado al que arriba* M, *luego de procesar todos los símbolos de la cadena* α, *partiendo desde el estado inicial* q_0, *es un estado de aceptación.*

$$AFD^{\Sigma} = \{M : M \text{ es un AFD con alfabeto de entrada } \Sigma\}$$

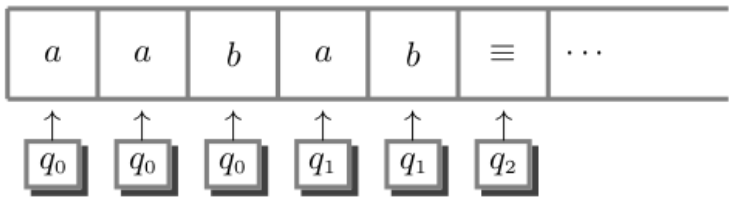
Ejemplo de un AFD

Sea $M_1 = (\Sigma, Q, q_0, F, \delta)$ con $\Sigma = \{a, b\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_0, q_2\}$ y δ definida de la siguiente manera:

δ	a	b
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_1	q_1

$$\delta(q_0, a) = q_0 \qquad \delta(q_0, b) = q_1$$
$$\delta(q_1, a) = q_1 \qquad \delta(q_1, b) = q_2$$
$$\delta(q_2, a) = q_1 \qquad \delta(q_2, b) = q_1.$$

La cadena $\alpha_1 =$ "*aabab*" es **aceptada** por M_1, porque luego de procesar todos los símblos de la cadena, el autómata arriba al estado $q_2 \in F$:



Mientras que la cadena $\alpha_2 =$ "*ababa*" es **rechazada** por M_1, pues el autómata arriba al estado $q_1 \notin F$:

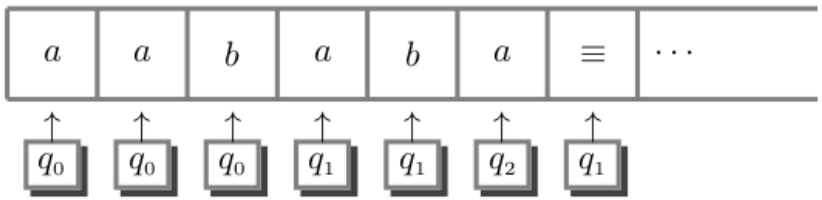
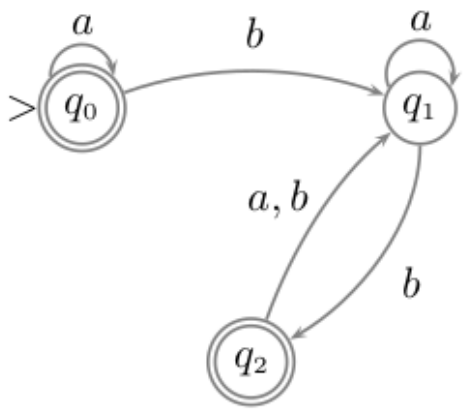


Diagrama de Transición de un AF

Representaremos un AF mediante un sistema de transición etiquetado, o sea, un grafo donde:

- Los nodos son los estados del AF.
- El nodo que se corresponde con el estado inicial se marca con una flecha pequeña a su lado (por convención q_0).
- Los nodos que se corresponden con los estados de aceptación se marcarán con un doble círculo.
- Existe una flecha desde el nodo "q" al nodo "q'", etiquetada con un símbolo "a", siempre que $\delta(q, a) = q'$.

Por ejemplo, para el autómata anterior M_1 su diagrama de transición sería:



Lenguaje Aceptado de un AFD

Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD, entonces definimos $\overrightarrow{\delta}(q, \alpha)$ como la función que devuelve el estado al que arriba M, luego de procesar la cadena α, partiendo del estado q:

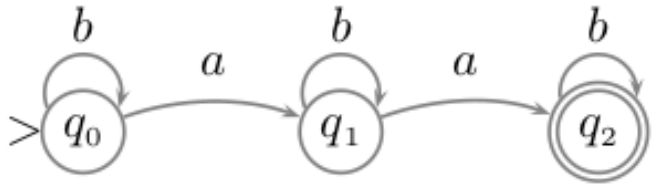
$\overrightarrow{\delta} : Q \times \Sigma^* \to Q$
$\overrightarrow{\delta}(q, \epsilon) = q$
$\overrightarrow{\delta}(q, a) = \delta(q, a)$
$\overrightarrow{\delta}(q, \alpha' a) = \delta(\overrightarrow{\delta}(q, \alpha'), a)$

Luego, definimos el **lenguaje aceptado** de M, denotado con $L(M)$, como el conjunto de todas cadenas aceptadas por M:

$$L(M) = \{\alpha \in \Sigma^* : \overrightarrow{\delta}(q_0, \alpha) \in F\}$$

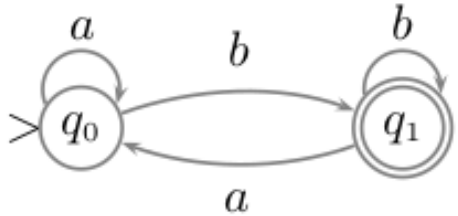
Ejemplos de L(M)

Por ejemplo, para el siguiente AFD M_1:



Tenemos que $L(M_1) = b^*ab^*ab^*$ (el lenguaje de todas las cadenas del alfabeto que tienen exactamente dos símbolos "a").

Para el AFD M_2:



Tenenemos que $L(M_2) = (a + b)^*b$ (el lenguaje de todas las cadenas del alfabeto que terminan con un símbolo "b").

Determinismo vs No-Determinismo

- **Determinismo**: dado un estado y un símbolo, existe un único estado al cual el autómata puede transicionar, por lo tanto, la traza de procesamiento de una cadena es única.

$$\delta(q, a) = q' \implies \overrightarrow{\delta}(q, \alpha) \text{ es un estado}$$

´

- **No-Determinismo**: dado un estado y un símbolo, existen varios estados a los cuales el autómata puede transicionar, por lo tanto, existen varias trazas de procesamiento para una misma cadena.

$$\delta(q, a) = \{q1, \ldots, q_k\} \implies \overrightarrow{\delta}(q, \alpha) \text{ es un conjunto de estados.}$$

(Para evitar confusiones, denotaremos con Δ las funciones de transición no-deterministas, mientras que con δ las deterministas)

Autómata Finito No-Determinístico (AFN)

Definición (AFN)

Un **Automata Finito No-Deterministico (AFN)** *es un 5-upla* $M = (\Sigma, Q, q_0, F, \Delta)$ *donde:*

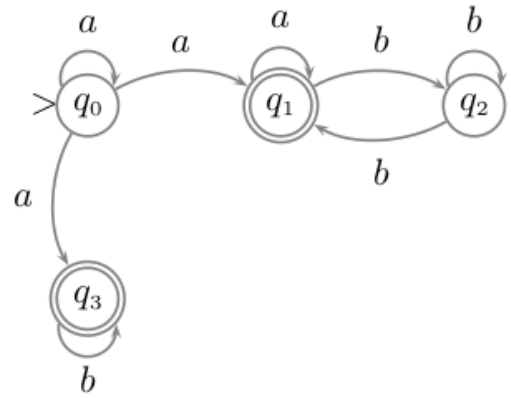
- Σ *es un alfabeto, llamado alfabeto de entrada.*
- Q *es un conjunto (no vacio y finito) de estados*
- $q_0 \in Q$, *llamado estado inicial*
- $F \subseteq Q$, *llamado conjunto de estados de aceptación o finales*
- $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$, *llamada función de transición, tal que,* $\Delta(q, a) = \{q_1, \ldots, q_k\}$ *significa que si el autómata se encuentra en el estado "q" y el cabezal lee un símbolo "a" en la cinta de entrada, entonces el autómata puede transicionar a cualquier estado* q_i.

Diremos que una cadena $\alpha \in \Sigma^*$ *es* **aceptada** *por el autómata* M, *si existe* **al menos una forma** *de procesar la cadena* α, *partiendo desde el estado inicial* q_0, *donde el estado al que arriba* M *es un estado de aceptación.*

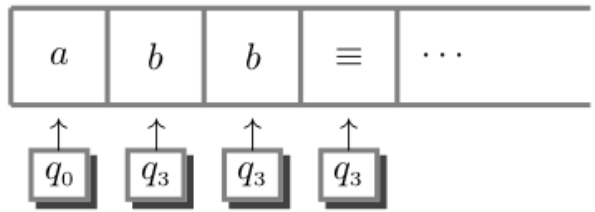
$$AFN^{\Sigma} = \{M : M \text{ es un AFN con alfabeto de entrada } \Sigma\}$$

Ejemplo de AFN

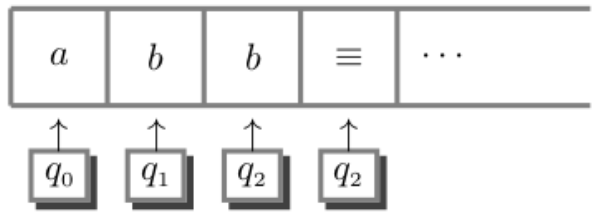
Sea M_1 el siguiente AFN:



La cadena "abb" tiene una traza de procesamiento de aceptación:



Y también, tiene una traza de procesamiento de rechazo:



Lenguaje Aceptado de un AFN

Sea $M = (\Sigma, Q, q_0, F, \Delta)$ un AFN, primero extendemos la definición de Δ a un conjunto de estados $S \subseteq Q$:

$$\hat{\Delta}(S, a) = \bigcup_{q \in S} \Delta(q, a)$$

Luego, definimos $\overrightarrow{\Delta}(q, a)$ como la función que devuelve el **conjunto de todos los estados posibles** al que puede arribar M luego de procesar la cadena α partiendo del estado q:

$$\overrightarrow{\Delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$$
$$\overrightarrow{\Delta}(q, \epsilon) = \{q\}$$
$$\overrightarrow{\Delta}(q, a) = \Delta(q, a)$$
$$\overrightarrow{\Delta}(q, \alpha' a) = \hat{\Delta}(\overrightarrow{\Delta}(q, \alpha'), a)$$

Para finalmente, definir el lenguaje aceptado de M como:

$$L(M) = \{\alpha \in \Sigma^* : \overrightarrow{\Delta}(q_0, \alpha) \cap F \neq \emptyset\}$$

Equivalencia entre AFD y AFN

Los AFD y los AFN son computacionalmente equivalentes.

Como toda equivalencia debemos probar una ida y una vuelta.

Teorema ($AFD^{\Sigma} \equiv AFN^{\Sigma}$)

1. *(Ida) Si* $M \in AFD^{\Sigma}$, *entonces* $\exists M' \in AFN^{\Sigma}$ *tq* $L(M') = L(M)$.
2. *(Vuelta) Si* $M \in AFN^{\Sigma}$, *entonces* $\exists M' \in AFD^{\Sigma}$ *tq* $L(M') = L(M)$.

Demo 1. Trivial, pues un AFD puede ser visto como un AFN sin usar su no-determinismo (i.e. basta con definir $\Delta'(q, a) = \{\delta(q, a)\}$).

Demo 2. Debemos probar que **siempre se puede determinizar un AFN preservando su lenguaje**. Y el truco consiste en que los estados del AFD son conjuntos de estados del AFN, de tal manera que si el AFD se encuentra en el estado $\{q1, \ldots, q_k\}$ luego de procesar un prefijo de la cadena, entonces el AFN se encuentra en algún q_i.

Sea $M = (\Sigma, Q, q_0, F, \Delta)$, entonces tomamos $M' = (\Sigma, Q', q_0', F', \delta')$ con:

$Q' = \mathcal{P}(Q)$
$q_0' = \{q_0\}$
$F' = \{S \subseteq Q : S \cap F \neq \emptyset\}$
$\delta'(S, a) = \hat{\Delta}(S, a) = \bigcup_{q \in S} \Delta(q, a)$

Claramente, vale que $\overrightarrow{\delta'}(\{q_0\}, \alpha) = \overrightarrow{\Delta}(q_0, \alpha)$ para toda α, por definición de M'.

Luego, $\alpha \in L(M)$
$\iff \exists q \in F \in$ tal que $q \in \overrightarrow{\Delta}(q_0, \alpha)$
$\iff \exists q \in F \in$ tal que $q \in \overrightarrow{\delta'}(\{q_0\}, \alpha)$
$\iff \overrightarrow{\delta'}(\{q_0\}, \alpha) \in F'$
$\iff \alpha \in L(M')$

$\therefore L(M) = L(M')$

Optimizando un AF por eliminación de estados

Hay estados que se pueden eliminar sin afectar el lenguaje aceptado de un AF:

- Estados que NO tienen un camino desde el estado inicial hasta ellos.
- Estados que NO tienen un camino desde ellos hasta un estado de aceptación.

Definición (Estados Alcanzables y Solubles)

Sea $M = (\Sigma, Q, q_0, F, \Delta)$ *un AFN, entonces:*

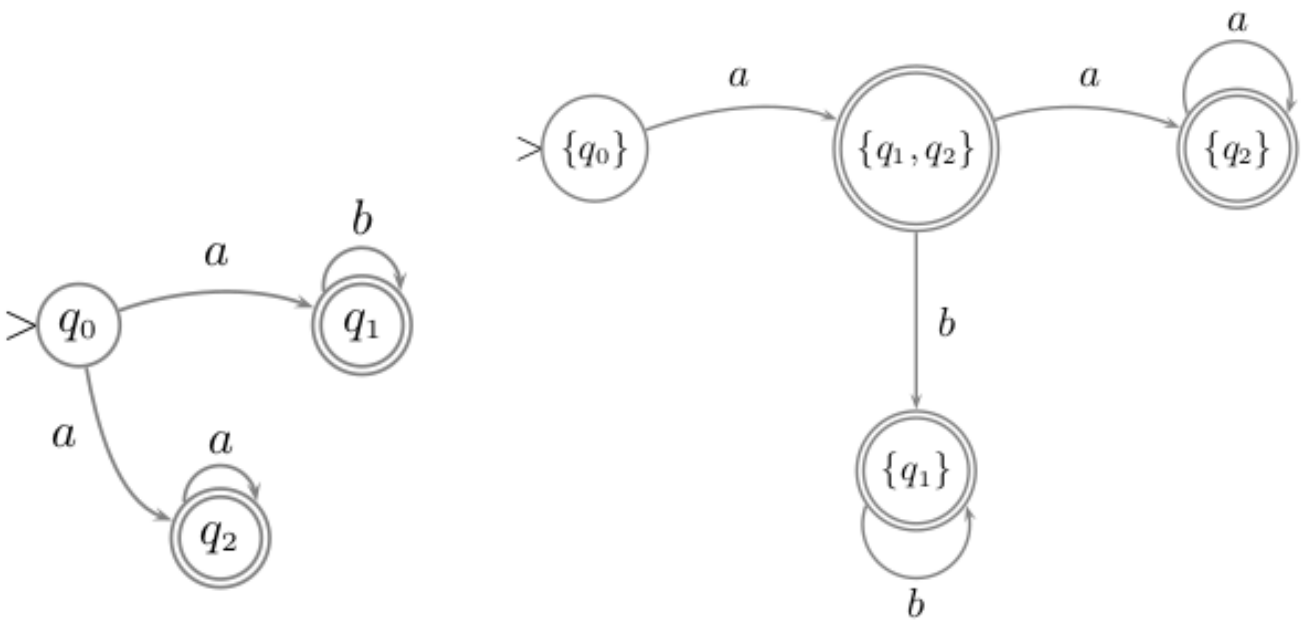
- *Un estado* $q \in Q$ *es* **alcanzable** *sii* $\exists \alpha \in \Sigma^*$ *tq* $q \in \overrightarrow{\Delta}(q_0, \alpha)$.
- *Un estado* $q \in Q$ *es* **soluble** *sii* $\exists \alpha \in \Sigma^*$ *tq* $\overrightarrow{\Delta}(q, \alpha) \cap F \neq \emptyset$.

Por lo tanto, podemos eliminar los estados **no alcanzables** y **no solubles** preservando completamente el lenguaje aceptado del autómata.

Determinizando un AFN

La última demo (2) nos brinda un algoritmo de determinización, pero, por lo general, con muchos estados no solubles y no alcanzables que se pueden eliminar sin afectar el lenguaje aceptado.

Por ejemplo, para el AFN que se muestra más a la izquierda, con lenguaje $ab^* + a^+$, aplicamos el algoritmo y luego de eliminar los estados no alcanzables y no solubles obtenemos a la derecha su AFD equivalente que acepta el mismo lenguaje:



Transiciones Espontáneas

- Permiten al autómata transicionar de estado, sin leer y desplazar el cabezal de la cinta de entrada.
- Una transición espontánea, si esta definida, puede tener lugar como no (por eso su nombre), en consecuencia, es en si misma una transición no-determinista.
- Se pueden pensar como una transición "gratis", en el sentido de que el autómata cambia de estado sin procesar o consumir un símbolo de la cinta de entrada.
- Las denotaremos con el símbolo ϵ, así:

$$\Delta(q, \epsilon) = \{q1, \ldots, q_k\}$$

significa que el autómata puede libremente transicionar de q a cualquier q_i sin leer símbolo alguno de la cinta de entrada (y sin desplazar su cabezal).

Definición (AFNϵ)

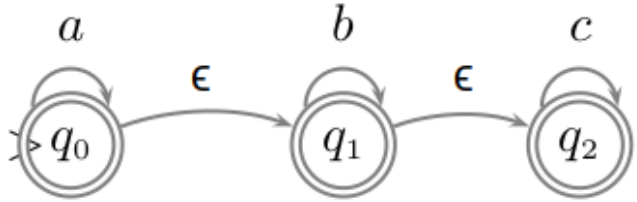
Un **Automata Finito No-Deterministico con transiciones espontáneas** *(AFNϵ) es un 5-upla* $M = (\Sigma, Q, q_0, F, \Delta)$ *donde:*

- Σ *es un alfabeto, llamado alfabeto de entrada*
- Q *es un conjunto (no vacio y finito) de estados*
- $q_0 \in Q$, *llamado estado inicial*
- $F \subseteq Q$, *llamado conjunto de estados de aceptación o finales*
- $\Delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$, *llamada función de transición.*

Diremos que una cadena $\alpha \in \Sigma^*$ *es* **aceptada** *por el autómata* M, *si existe* **al menos una forma** *de procesar la cadena* α, *partiendo desde el estado inicial* q_0, *donde el estado al que arriba* M *es un estado de aceptación.*

$$AFN\epsilon^{\Sigma} = \{M : M \text{ es un } AFN\epsilon \text{ con alfabeto de entrada } \Sigma\}$$

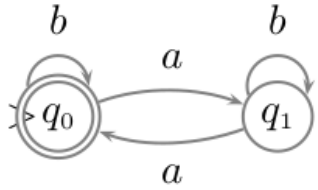
El AFNϵ que acepta el lenguaje $a^*b^*c^*$:



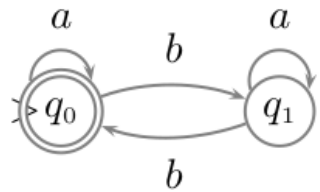
Las transiciones espontáneas son muy útiles ya que permiten muy fácilmente realizar empalmes entre autómatas!!!

Combinando Autómatas

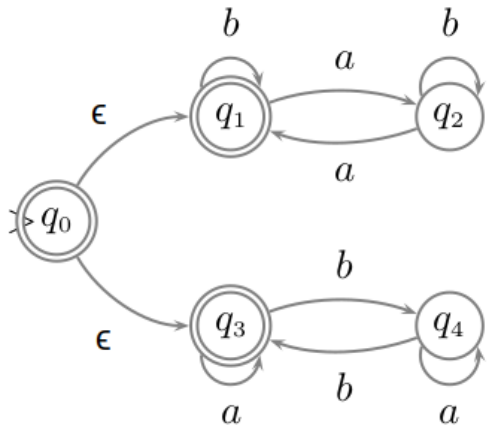
AFD que acepta el lenguaje de las cadenas con un número par de a's:



AFD que acepta el lenguaje de las cadenas con un número par de b's:



AFNϵ que acepta el lenguaje de las cadenas con un número par de a's ó b's:



Equivalencia entre AFN y AFNϵ

Los AFN y AFNϵ son computacionalmente equivalentes.

Teorema ($AFN^{\Sigma} \equiv AFN\epsilon^{\Sigma}$)

1. *(Ida) Si* $M \in AFN^{\Sigma}$, *entonces* $\exists M' \in AFN\epsilon^{\Sigma}$ *tq* $L(M') = L(M)$.
2. *(Vuelta) Si* $M \in AFN\epsilon^{\Sigma}$, *entonces* $\exists M' \in AFN^{\Sigma}$ *tq* $L(M') = L(M)$.

Demo 1. Trivial, pues un AFN puede ser visto como un AFNϵ sin utilizar transiciones espontáneas.
Demo 2. Debemos probar que **siempre se puede eliminar las transiciones espontáneas sin afectar el lenguaje**. El truco consiste en introducir transiciones que simulen los mismos efectos de las espontáneas. Para esto, debemos introducir primero la noción de **clausura de un estado**.

Clausura de un Estado

La **clausura** de un estado q, denotado con $[q]$, es el conjunto de estados al que puedo arribar desde q, utilizando solamente transiciones espontáneas:

$$q \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} q_2 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} q_k \Longrightarrow q_i \in [q]$$

Definimos formalmente la clausura de un estado q:

$$[q] = \{q' \in Q : q' \in \overrightarrow{\Delta}(q, \epsilon)\}$$

Extendemos la definición de clausura para conjunto de estados $S \subseteq Q$:

$$[S] = \bigcup_{q \in S} [q]$$

Para todo q, se cumple que:

1. $q \in [q]$
2. $[[q]] = [q]$

Ahora si probemos la vuelta (2).

Sea $M = (\Sigma, Q, q_0, F, \Delta)$ el AFNϵ, entonces podemos tomar el AFN $M' = (\Sigma, Q', q_0', F', \Delta')$ donde:

Se preservan todos los estados y el estado inicial del AFNϵ:

$Q' = Q$
$q_0' = q_0$

Los estados finales son aquellos tal que su clausura contiene al menos un estado de aceptación del AFNϵ:

$F' = \{q \in Q : [q] \cap F \neq \emptyset\}$

Y por último, intuitivamente si en el AFNϵ tenemos un camino $q \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} q_1 \xrightarrow{a} q_2 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} q'$, entonces en el AFN definimos la transición $q \xrightarrow{a} q'$:

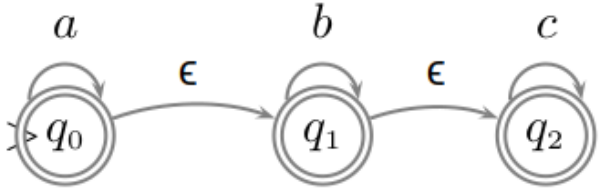
$\Delta'(q, a) = [\hat{\Delta}([q], a)]$

Luego, $L(M) = L(M')$.

Algoritmo de Eliminacion de Transiciones Espontaneas

La prueba del iniciso 2 nos brinda un algoritmo para eliminar transiciones espontáneas.

Por ejemplo, sea M el siguiente AFNϵ con L(M) = $a^*b^*c^*$.



Obtengamos el AFN equivalente eliminando sus transiciones espontáneas.

Calculamos las clausuras de los estados:
$[q_0] = \{q_0, q_1, q_2\}$
$[q_1] = \{q_1, q_2\}$
$[q_2] = \{q_2\}$

Calculamos los estados de aceptación:
$F' = \{q_0, q_1, q_2\}$

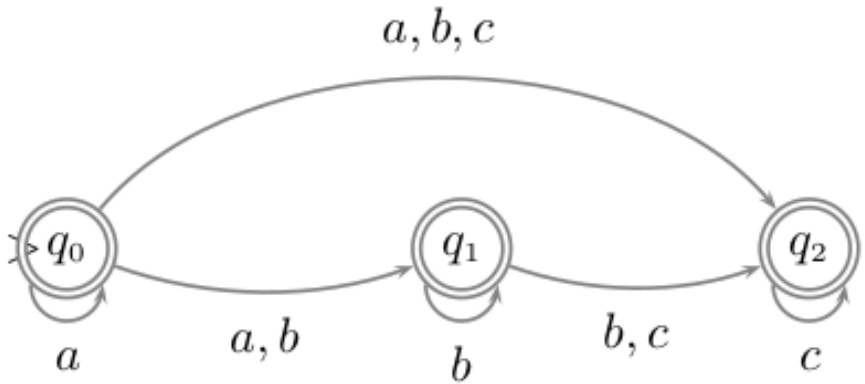
Y por último, calculamos la función de transición:

$$\Delta'(q_0, a) = [\hat{\Delta}([q_0], a)] = [\hat{\Delta}(\{q_0, q_1, q_2\}, a)] = [\{q_0\}] = \{q_0, q_1, q_2\}$$
$$\Delta'(q_0, b) = [\hat{\Delta}([q_0], b)] = [\hat{\Delta}(\{q_0, q_1, q_2\}, b)] = [\{q_1\}] = \{q_1, q_2\}$$
$$\Delta'(q_0, c) = [\hat{\Delta}([q_0], c)] = [\hat{\Delta}(\{q_0, q_1, q_2\}, c)] = [\{q_2\}] = \{q_2\}$$

$$\Delta'(q_1, a) = [\hat{\Delta}([q_1], a)] = [\hat{\Delta}(\{q_1, q_2\}, a)] = [\emptyset] = \{q_0, q_1, q_2\}$$
$$\Delta'(q_1, b) = [\hat{\Delta}([q_1], b)] = [\hat{\Delta}(\{q_1, q_2\}, b)] = [\{q_1\}] = \{q_1, q_2\}$$
$$\Delta'(q_1, c) = [\hat{\Delta}([q_1], c)] = [\hat{\Delta}(\{q_1, q_2\}, c)] = [\{q_2\}] = \{q_2\}$$

$$\Delta'(q_2, a) = [\hat{\Delta}([q_2], a)] = [\hat{\Delta}(\{q_2\}, a)] = [\emptyset] = \emptyset$$
$$\Delta'(q_2, b) = [\hat{\Delta}([q_2], b)] = [\hat{\Delta}(\{q_2\}, b)] = [\emptyset] = \emptyset$$
$$\Delta'(q_2, c) = [\hat{\Delta}([q_2], c)] = [\hat{\Delta}(\{q_2\}, c)] = [\{q_2\}] = \{q_2\}$$

Obteniendo finalmente el siguiente AFN M' equivalente pues $L(M') = a^* b^* c^*$.



Resumen

- Presentamos tres tipos de autómatas finitos (AFD, AFN, AFNϵ) y probamos que son computacionalmente equivalentes :

$$AFD^{\Sigma} \equiv AFN^{\Sigma} \text{ y } AFN^{\Sigma} \equiv AFN\epsilon^{\Sigma} \implies AFD^{\Sigma} \equiv AFN\epsilon^{\Sigma}$$

- En la praćtica, convertimos un AFNϵ a su AFD equivalente, aplicando primero el algoritmo de eliminacion de transiciones espontáneas, y luego, el algoritmo de determinización.
- Por ser equivalentes, hablaremos de autómata finito (AF) sin especificar su tipo:

$$AF^{\Sigma} = AFD^{\Sigma} \cup AFN^{\Sigma} \cup AFN\epsilon^{\Sigma}$$

- Ahora estamos en condiciones de probar en la próxima clase que los AF caracterizan a los lenguajes regulares:

$$LR^{\Sigma} \equiv AF^{\Sigma}.$$

Equivalencia entre Lenguajes Regulares y Autómatas Finitos (Teorema de Kleene)

Introducción a la Lógica y la Computación (3era Parte)

Docentes: Badano, Bustos, Costamagna, Tellechea, Zigaran

Año 2024

Equivalencia entre Lenguajes Regulares y Autómatas Finitos

Un lenguaje es regular si y solo si es aceptado por un AF.

"Una interpretación posible de este resultado es que los AF son las máquinas secuenciales que caracterizan a los lenguajes regulares."

- Ya probamos que $LR^{\Sigma} \equiv ER^{\Sigma}$ y $AFD^{\Sigma} \equiv AFN^{\Sigma} \equiv AFN\epsilon^{\Sigma}$.
- Uniremos esta dos cadenas de equivalencia probando que $ER^{\Sigma} \equiv AF^{\Sigma}$.

Teorema de Kleene ($ER^{\Sigma} \equiv AF^{\Sigma}$)

1. *(ida) Si* $e \in ER^{\Sigma}$, *entonces* $\exists M \in AFN\epsilon^{\Sigma}$ *tq* $L(M) = L(e)$.
2. *(vuelta) Si* $M \in AFN^{\Sigma}$, *entonces* $\exists e \in ER^{\Sigma}$ *tq* $L(M) = L(e)$.

Para probar el inciso 1, notar que:

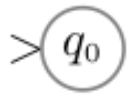
- Un AF puede ser visto como una estructura recursiva: es un estado conectado con n sub-autómatas.
- Por lo tanto, podemos construir un AF recursivamente acompañando la recursividad de la expresiones regulares.

Demo 1.

Hacemos inducción en la forma de la expresión regular e.

Casos Bases:

Si $e = \emptyset$, entonces tomamos M:

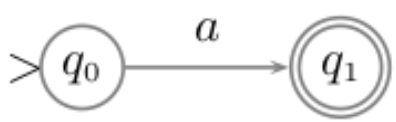


y luego, $L(M) = \emptyset = L(\emptyset)$.

Si $e = \epsilon$, entonces tomamos M:

y luego, $L(M) = \{\epsilon\} = L(\epsilon)$.

Si $e = a$, entonces tomamos M:

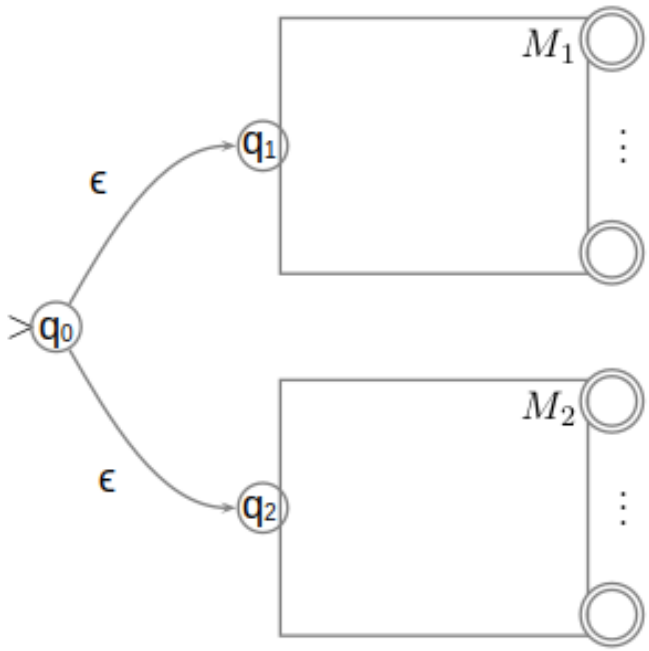


y luego, $L(M) = \{a\} = L(a)$.

Casos Recursivos:

Si $e = e_1 + e_2$, por HI, existen $M_1 = (\Sigma, Q_1, q_1, F_1, \Delta_1)$ y $M_2 = (\Sigma, Q_2, q_2, F_2, \Delta_2)$ $AFN\epsilon$ tq $L(M_1) = L(e_1)$ y $L(M_2) = L(e_2)$ $\therefore$ podemos tomar $M = (\Sigma, Q, q_0, F, \Delta)$ AFNϵ donde:

$Q = Q_1 \cup Q_2 \cup \{q_0\}$ con $q_0 \notin Q_1, Q_2$
$F = F_1 \cup F_2$
$\Delta = \Delta_1 \cup \Delta_2 \cup \{((q_0, \epsilon), \{q_1, q_2\})\}$



Luego, $L(M) = L(M_1) \cup L(M_2) = L(e_1) \cup L(e_2) = L(e_1 + e_2)$.

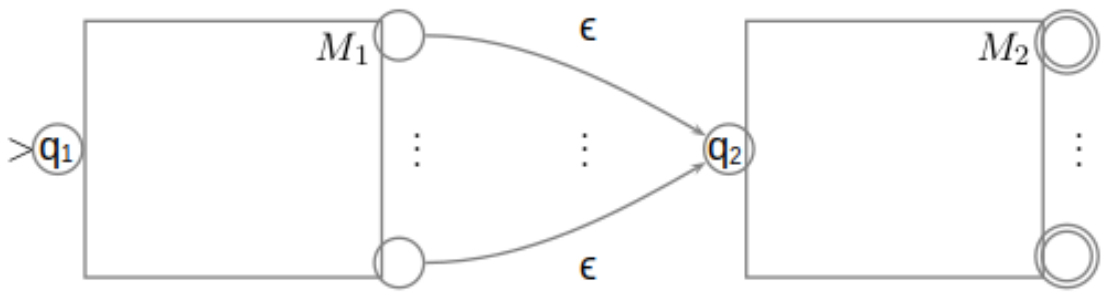
Si $e = e_1 e_2$, por HI, existen $M_1 = (\Sigma, Q_1, q_1, F_1, \Delta_1)$ y $M_2 = (\Sigma, Q_2, q_2, F_2, \Delta_2)$ $AFN\epsilon$ tq $L(M_1) = L(e_1)$ y $L(M_2) = L(e_2)$ $\therefore$ podemos tomar $M = (\Sigma, Q, q_0, F, \Delta)$ AFNϵ donde:

$Q = Q_1 \cup Q_2$

$q_0 = q_1$

$F = F_2$

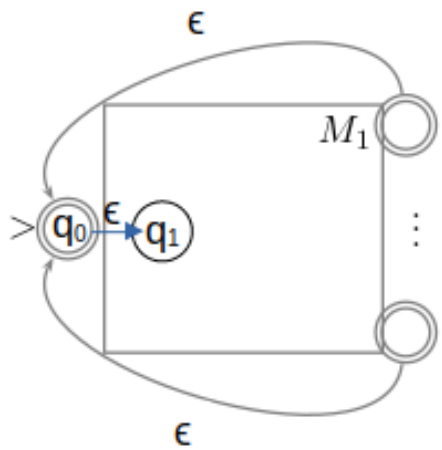
$\Delta = (\Delta_1 \cup \Delta_2) - \{((q, \epsilon), \Delta_1(q, \epsilon)) : q \in F_1\}$
$\cup \{((q, \epsilon), \Delta_1(q, \epsilon) \cup \{q_2\}) : q \in F_1\}$



Luego, $L(M) = L(M_1)L(M_2) = L(e_1)L(e_2) = L(e_1 e_2)$.

Si $e = e_1{}^*$, por HI, existe $M_1 = (\Sigma, Q_1, q_1, F_1, \Delta_1)$ $AFN\epsilon$ tq
$L(M_1) = L(e_1)$ $\therefore$ podemos tomar $M = (\Sigma, Q, q_0, F, \Delta)$ AFNϵ donde:

$Q = Q_1 \cup \{q_0\}$, con $q_0 \notin Q_1$
$F = F_1 \cup \{q_0\}$
$\Delta = \Delta_1 - \{((q, \epsilon), \Delta_1(q, \epsilon)) : q \in F_1\}$
$\cup\{((q, \epsilon), \Delta_1(q, \epsilon) \cup \{q_0\}) : q \in F_1\}$
$\cup\{((q0, \epsilon), \{q_1\})\}$



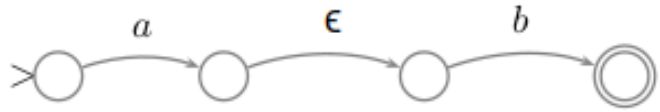
Luego, $L(M) = (L(M_1))^* = (L(e_1))^* = L(e_1{}^*)$.

Algoritmo de Kleene (ida)

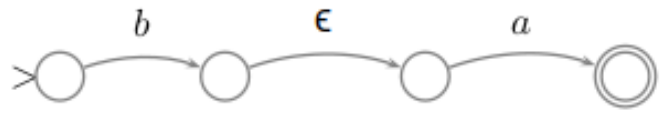
La demo anterior nos brinda un algoritmo recursivo que dada una expresión regular, devuelve un AF equivalente.

Por ejemplo, para la expresión regular $ab + ba$, hacemos:

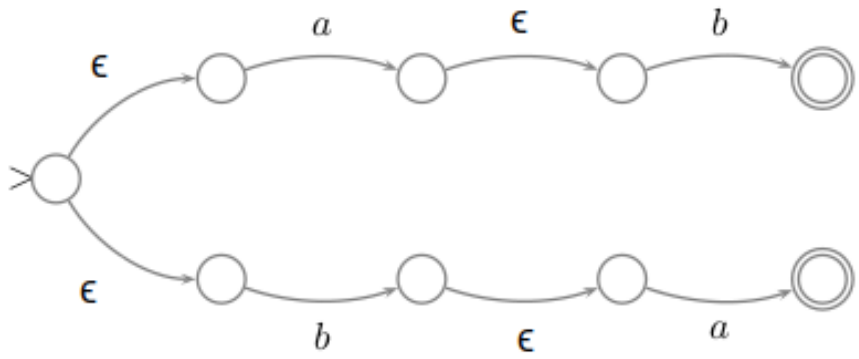
1) AF que acepta ab:



2) AF que acepta ba:



3) AF que acepta $ab + ba$:



Teorema de Kleene (vuelta)

Para probar la vuelta del Teorema de Kleene (2) necesitamos enunciar el siguiente lema:

Lema (de Arden)

Si A, B son lenguajes y $\epsilon \notin A$, entonces la ecuación $X = AX \cup B$ tiene solución única y es $X = A^*B$.

Notar que si A y B son regulares, entonces:
1) X es regular
2) A, B y X pueden ser denotados por expresiones regulares

Entonces:

$X = aX + b^*ab$ tiene solución única $X = a^*b^*ab$.

$X = a^2X + b^+X + ab = (a^2 + b^+)X + ab$ tiene solución única $X = (a^2 + b^+)^*ab$.

$X = ab^2X + aX + a^*b + b^*a = (ab^2 + a)X + (a^*b + b^*a)$ tiene solución única $X = (ab^2 + a)^*(a^*b + b^*a)$.

Teorema de Kleene (vuelta)

Demo 2. Sea $M = (\Sigma, Q, q_0, F, \delta)$ AFN con $Q = \{q_0, q_1, \ldots, q_n\}$, definimos $M_i = (\Sigma, Q, q_i, F, \Delta)$ el AFN que se obtiene a partir de M cambiando únicamente el estado inicial de M por q_i.

Y definimos $L(M_i) = X_i$, por lo tanto, $L(M) = X_0$.

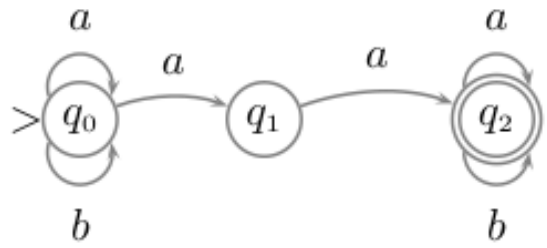
Cada X_i se puede expresar con la siguiente ecuación:

$$X_i = \begin{cases} \sum_{q_j \in \Delta(q_i, a)} aX_j & \text{si } q_i \notin F \\ \sum_{q_j \in \Delta(q_i, a)} aX_j + \epsilon & \text{si } q_i \in F \end{cases}$$

Por lo tanto, tenemos un sistema de $(n+1)$ ecuaciones por $(n+1)$ incógnitas al cual le aplicaremos el leman de Arden sucesivamente de la siguiente manera:

- Aplicamos Arden a la ecuación de X_n para obtener su valor y reemplazarlo en el resto de las ecuaciones reduciéndolo a un sistema $n \times n$.
- Repetimos lo anterior para la ecuación de X_{n-1}, y asi sucesivamente, hasta obtener un sistema de 1x1 con la ecuación correspondiente a X_0.
- Aplicamos por última vez Arden, obteniendo la solución final que buscabamos, pues $X_0 = L(M)$.

Por ejemplo, consideremos el siguiente AFN M:



El sistema de ecuaciones que determina el automata M es:

$$\begin{cases} X_0 = aX_0 + bX_0 + aX_1 & (1) \\ X_1 = aX_2 & (2) \\ X_2 = aX_2 + bX_2 + \epsilon & (3) \end{cases}$$

Aplicamos Arden en la ecuacion (3):

$X_2 = aX_2 + bX_2 + \epsilon = (a+b)X_2 + \epsilon \therefore$
$X_2 = (a+b)^*$

Reemplazamos el valor obtenido en todo el sistema:

$$\begin{cases} X_0 = aX_0 + bX_0 + aX_1 & (1) \\ X_1 = a(a+b)^* & (2) \end{cases}$$

No es necesario aplicar Arden en la ecuacion (2) pues:

$X_1 = a(a+b)^*$

Reemplazamos el valor de X_1 obtenido en el resto del sistema:

$$\{X_0 = aX_0 + bX_0 + a^2(a+b)^* \quad (1)$$

Aplicamos Arden por última vez a la ecuación (1):

$X_0 = aX_0 + bX_0 + a^2(a+b)^* = (a+b)X_0 + a^2(a+b)^* \therefore$
$X_0 = (a+b)^* a^2(a+b)^*$

Y esta última solución de X_0 es la expresión regular que denota al lenguaje aceptado por el autómata M, por lo tanto:

$L(M) = (a+b)^* a^2(a+b)^*$

Resumen

- Probamos que $ER^{\Sigma} \equiv AF^{\Sigma}$ (Teo. de Kleene)
- Por transitividad, tenemos toda la cadena de equivalencias probada:

$$LR^{\Sigma} \equiv ER^{\Sigma} \equiv AFD^{\Sigma} \equiv AFN^{\Sigma} \equiv AFN\epsilon^{\Sigma}$$

- Por lo tanto, podemos asegurar que un lenguaje es regular si y solo si es aceptado por un AF.
- En consecuencia, tenemos una nueva herramienta para probar regularidad de un lenguaje simplemente dando el autómata finito que lo acepta.
- Podemos pensar este resultado como que los AF's son las máquinas secuenciales que caracterizan a los lenguajes regulares.
- En la próxima clase, presentaremos un nuevo modelo computacional que también caracteriza a los lenguajes regulares, las "gramáticas regulares".

Gramáticas Regulares

Introducción a la Logica y la Computación (3era Parte)

Docentes: Badano, Bustos, Costamagna, Tellechea, Zigaran

Año 2024

Gramáticas de Chomsky

- Hasta aquí, hemos probado que los lenguajes regulares, las expresiones regulares y los autómatas finitos son equivalentes:

$$LR^{\Sigma} \equiv ER^{\Sigma} \equiv AF^{\Sigma}$$

- Vamos a presentar el último modelo computacional que caracteriza a los lenguajes regulares, las llamadas **gramáticas regulares**.
- Las gramáticas fueron el modelo matemático propuesto originalmente por Chomsky para caracterizar cada una de las clases de lenguajes de su jerarquía:

Lenguaje Regular	$\leftrightarrow$	Gramática Regular
Lenguaje Indep. Contexto	$\leftrightarrow$	Gramática Indep. Contexto
Lenguaje Sens. Contexto	$\leftrightarrow$	Gramática Sens. de Contexto
Lenguaje Rec. Enumerables	$\leftrightarrow$	Gramática Sin Restricciones

Gramática de Chomsky

- Definimos primero la versión más general del modelo, denominado gramática sin restricciones.
- Y luego, le impondremos una fuerte restricción dando lugar a la llamada gramática regular.

Definición (Gramática Sin Restricciones)

Una gramática es una 4-upla $G = (V, \Sigma, S, P)$, *con* $V \cap \Sigma = \emptyset$, *donde:*

- V *es un conjunto finito de* **símbolos no terminales**, *llamado conjunto de variables.*
- Σ *es un conjunto finito de* **símbolos terminales**.
- $S \in V$, *llamada* **variable inicial**.
- P *es un conjunto finito de* **producciones** *de la forma* $\alpha \to \beta$ *tal que* $\alpha, \beta \in (V \cup \Sigma)^*$ *y* $\alpha \neq \epsilon$.

Cómo Funciona una Gramática?

- Una gramática permite generar o derivar cadenas de símbolos terminales mediante una sucesión de reemplazos sintácticos definidos en su conjunto de producciones.
- La producción $\alpha \rightarrow \beta$ significa que podemos **sustituir** la ocurrencia de la cadena α por la cadena β.
- Por ejemplo, sea $G_1 = (\{S\}, \{a, b\}, S, \{S \rightarrow aS, S \rightarrow \epsilon\})$ una gramática, podemos derivar la cadena "*aaa*" en G_1 de la siguiente manera:

$$S \underset{G_1}{\Longrightarrow} aS \underset{G_1}{\Longrightarrow} aaS \underset{G_1}{\Longrightarrow} aaaS \underset{G_1}{\Longrightarrow} aaa$$

- Todas la cadenas que se pueden derivar en una gramática definen el lenguaje de la gramática.
- Por legibilidad, describimos las gramáticas listando sus producciones separadas por un "|":

$$G_1 : S \rightarrow aS|\epsilon$$

Derivación de una Cadena

Definimos formalmente la noción de 1-paso de derivación:

$$\alpha \underset{G}{\Longrightarrow} \beta \text{ sii } \exists \alpha_1, \alpha_2, \alpha_2', \alpha_3 \in (V \cup \Sigma)^* \text{ tq } \alpha = \alpha_1\alpha_2\alpha_3$$
$$\beta = \alpha_1\alpha_2'\alpha_3$$
$$\alpha_2 \to \alpha_2' \in P$$

Y luego, n-pasos de derivación:

$$\alpha \overset{n}{\underset{G}{\Longrightarrow}} \beta \text{ sii } \exists \alpha_1, \ldots, \alpha_{n+1} \in (V \cup \Sigma)^* \text{ tq}$$
$$\alpha = \alpha_1 \underset{G}{\Longrightarrow} \ldots \underset{G}{\Longrightarrow} \alpha_{n+1} = \beta$$

Y finalmente, escribimos $\alpha \overset{*}{\underset{G}{\Longrightarrow}} \beta$ sii $\alpha \overset{n}{\underset{G}{\Longrightarrow}} \beta$ para algún $n \in \mathbb{N}$.

Lenguaje Generado de una Gramática

Son todas las cadenas de terminales que pueden ser derivadas en la gramática a partir de su variable inicial:

$$L(G) = \{\alpha \in \Sigma^* : S \overset{*}{\underset{G}{\Longrightarrow}} \alpha\}$$

Para la gramática anterior:

$$G_1 : S \to aS|\epsilon$$

Tenemos que $L(G_1) = \{a^n : n \in \mathbb{N}\}$.

Sea G_2 la siguiente gramática:

$$G_2 : S \to \epsilon|bS|aA$$
$$A \to aS|bA$$

Tenemos que $L(G_2) = \{\alpha \in \Sigma_2^* : |\alpha|_a \text{ es par}\}$.

Sea G_3 la siguiente gramática:

$$G_3 : S \rightarrow aSb|\epsilon$$

Tenemos que $L(G_3) = \{a^m b^m : m \in \mathbb{N}\}$.

Gramáticas Regulares

Surgen de imponer una fuerte restricción en la forma de las producciones $\alpha \to \beta \in P$:

- α sólo puede ser una única variable.
- β sólo puede ser una cadena de símbolos terminales seguida de a lo sumo una variable o viceversa.

Definición (Gramática Regular)

Una gramática $G = (V, \Sigma, S, P)$ *es* **Lineal por Derecha** *si y solo si todas sus producciones son de la forma* $A \to \alpha B$ *o* $A \to \alpha$ *con* $\alpha \in \Sigma^*$ *y* $A, B \in V$. *Análogamente, una gramática* $G = (V, \Sigma, S, P)$ *es* **Lineal por Izquierda** *si y solo si todas sus producciones son de la forma* $A \to B\alpha$ *o* $A \to \alpha$ *con* $\alpha \in \Sigma^*$ *y* $A, B \in V$.

Una gramática $G = (V, \Sigma, S, P)$ *es una* **Gramática Regular (GR)** *si y solo si es una gramática lineal por derecha o una gramática lineal por izquierda.*

$$GR^{\Sigma} = \{G : G \text{ es } GR \text{ con simbolos terminales } \Sigma\}$$

Ejemplos de Gramáticas Regulares

De los ejemplos anteriores, G_1 y G_2 son gramáticas regulares, mientras que G_3 no lo es pues la producción $S \to aSb$ no es lineal.

Sea G_4 la gramática regular:

$$\begin{aligned} G_4 : S &\to aS|B \\ B &\to bB|\epsilon \end{aligned}$$

Tenemos que $L(G_4) = \{a^n b^m : n, m \in \mathbb{N}\} = a^* b^*$

Sea G_5 la gramática regular:

$$\begin{aligned} G_5 : S &\to aS|bS|A \\ A &\to aA|a \end{aligned}$$

Tenemos que $L(G_5) = (a + b)^* a^+$

Probando por Inducción que L(G) = L

Sea $L = \{\alpha \in \Sigma^* : |\alpha| \ par\}$ y sea G la gramática regular:

$$G : S \to aaS|abS|baS|bbS|\epsilon$$

Tenemos que $L(G) = L$.

Pero como podemos asegurar con certeza matemática que efectivamente $L(G) = L$? Debemos ver que vale la inclusión mutua $L(G) \subseteq L$ y $L \subseteq L(G)$. Y lo probaremos por inducción!

Veamos que $L(G) \subseteq L$.
Sea $\alpha \in L(G) \therefore S \overset{*}{\underset{G}{\Longrightarrow}} \alpha$, con $\alpha \in \Sigma^* \therefore S \overset{n}{\underset{G}{\Longrightarrow}} \alpha$, para algún $n \geq 1$.
Hacemos inducción en n (i.e, en la cantidad de pasos de la derivación).

HI: $S \overset{n}{\underset{G}{\Longrightarrow}} \alpha$, entonces $|\alpha| \in L$ (o sea, $|\alpha|$ par).

Caso base $n = 1$:
Si $S \overset{1}{\underset{G}{\Longrightarrow}} \alpha$, entonces $\alpha = \epsilon$ y $|\epsilon| = 0$ y es par $\therefore \epsilon \in L$.

Caso inductivo $n+1$:
Si $S \overset{n+1}{\underset{G}{\Longrightarrow}} \alpha \therefore S \overset{1}{\underset{G}{\Longrightarrow}} x_1x_2S \overset{n}{\underset{G}{\Longrightarrow}} \alpha$, con $x_1, x_2 \in \{a, b\} \therefore \alpha = x_1x_2\alpha'$, con $\alpha' \in \Sigma^* \therefore S \overset{n}{\underset{G}{\Longrightarrow}} \alpha'$, luego por HI, $\alpha' \in L \therefore |\alpha'|$ es par
$\therefore |\alpha| = |x_1x_2\alpha'| = 2 + |\alpha'|$ es par $\therefore \alpha \in L$.

Veamos que $L \subseteq L(G)$.
Sea $\alpha \in L$ y sea $n = |\alpha|$, con $n \geq 0$.

Hacemos inducción en n (i.e, en la longitud de la cadena).

HI: $\alpha \in L$ y $|\alpha| = n$, entonces $S \overset{*}{\underset{G}{\Longrightarrow}} \alpha$ (o sea, $\alpha \in L(G)$)

Caso base $n = 0$:
Si $0 = |\alpha|$, entonces $\alpha = \epsilon$, entonces tomamos $S \overset{1}{\underset{G}{\Longrightarrow}} \epsilon \therefore \epsilon \in L(G)$.

Caso inductivo $n+2$:
Si $n+2 = |\alpha| \therefore \alpha = x_1x_2\alpha'$, con $\alpha' \in \Sigma^*$, $|\alpha'|$ par y $x_1, x_2 \in \{a, b\}$
$\therefore \alpha' \in L$, luego por HI, $\alpha' \in L(G) \therefore S \overset{*}{\underset{G}{\Longrightarrow}} \alpha'$, entonces tomamos
$S \overset{1}{\underset{G}{\Longrightarrow}} x_1x_2S \overset{*}{\underset{G}{\Longrightarrow}} x_1x_2\alpha' = \alpha \therefore \alpha \in L(G)$.

(Este no fue el caso, pero en general, en aquellas gramáticas que utilizan más variables que la variable inicial S, debemos fortalecer la hipótesis inductiva!!!).

Equivalencia entre Lenguajes Regulares y Gramáticas Regulares

- Queremos probar que los lenguajes regulares y las gramáticas regulares son equivalentes, $LR^{\Sigma} \equiv GR^{\Sigma}$.
- Para probarlo de manera sencilla necesitamos primero introducir el concepto de gramática regular de paso único (GRPU).
- Probaremos que las GR y las GRPU son equivalentes.
- Por último, probaremos que los AF y las GRPU son equivalentes, y asi, por transitividad, habremos demostrado que $LR^{\Sigma} \equiv GR^{\Sigma}$.

Gramáticas Regulares de Paso Unico (GRPU)

- Surgen de imponer una restricción aún más fuerte a la forma de las producciones de una gramática regular.
- Por lo tanto, son un caso particular de gramáticas regulares.

Definición (Gramática Regular de Paso Unico)

Una gramática $G = (V, \Sigma, S, P)$ *es una* **Gramática Regular de Paso Unico** *(GRPU) si y solo si todas sus producciones son de la forma* $A \to aB$ *o* $A \to B$ *o* $A \to \epsilon$, *con* $a \in \Sigma$ *y* $A, B \in V$.

$$GRPU^{\Sigma} = \{G : G \text{ es } GRPU \text{ con simbolos terminales } \Sigma\}$$

De las anteriores, G_1, G_2, G_4 y G_5 son gramáticas regulares de paso unico.

Mientras que la gramática anterior G con $L(G) = \{\alpha \in \Sigma^* : |\alpha| \text{ par}\}$ no es de paso único pero si es regular:

$$G : S \to aaS|abS|baS|bbS|\epsilon$$

GRPU vs GR

En una gramática regular podemos derivar con la aplicación de una sola producción una cadena α con k símbolos:

$S \overset{1}{\Rightarrow} \alpha A$, con $\alpha = a_1 \dots a_k$

Mientras que en una gramática regular de paso único derivamos α pero aplicando k producciones:

$S \overset{1}{\Rightarrow} a_1 A_1 \overset{1}{\Rightarrow} a_1 a_2 A_2 \dots \overset{1}{\Rightarrow} a_1 a_2 \dots a_k Ak \overset{1}{\Rightarrow} a_1 a_2 \dots a_k A = \alpha A \ \therefore$

$S \overset{k}{\Rightarrow} \alpha A$

Por lo tanto, las GRPU pueden generar las mismas cadenas que una GR pero símbolo a símbolo.

Las GRPU pueden ser vistas como la versión "bit a bit" de las GR!!!

Equivalencia entre GRPU y GR

Las gramáticas regulares y las gramáticas regulares de paso único generan exactamente los mismos lenguajes.

Teorema ($GRPU^{\Sigma} \equiv GR^{\Sigma}$)

1. *(ida) Si* $G \in GRPU^{\Sigma}$, *entonces* $\exists G' \in GR^{\Sigma}$ *tal que* $L(G') = L(G)$.
2. *(vuelta) Si* $G \in GR^{\Sigma}$, *entonces* $\exists G' \in GRPU^{\Sigma}$ *tal que* $L(G') = L(G)$.

Demo 1. Trivial, pues una GRPU es en particular una GR.

Demo 2. Debemos reemplazar con producciones de paso único aquellas producciones de G que no lo son (y además, preservar las que si lo son).

Sean P y P' las producciones de G y G', respectivamente.

Si $(A \to a_1 \dots a_k B) \in P$, con $k \geq 2$, entonces en P' hacemos:

$$\begin{aligned} A &\to A_{a_1} \\ A_{a_1} &\to a_1 A_{a_2} \\ A_{a_2} &\to a_2 A_{a_3} \\ &\vdots \\ A_{a_k} &\to a_k A_{a_{k+1}} \\ A_{a_{k+1}} &\to B \end{aligned}$$

Si $(A \to a_1 \dots a_k) \in P$, con $k \geq 2$, entonces en P' hacemos:

$$\begin{aligned} A &\to A_{a_1} \\ A_{a_1} &\to a_1 A_{a_2} \\ A_{a_2} &\to a_2 A_{a_3} \\ &\vdots \\ A_{a_k} &\to a_k A_{a_{k+1}} \\ A_{a_{k+1}} &\to \epsilon \end{aligned}$$

Claramente, G' es GRPU, y además, $\alpha \in L(G)$ sii $\alpha \in L(G') \therefore L(G) = L(G')$.

Probando que $LR^{\Sigma} \equiv GR^{\Sigma}$

Un lenguaje es regular si y solo si es generado por una gramática regular.

"Una interpretación posible de este resultado es que las GR son el modelo computacional, además de los AF, que caracterizan a los lenguajes regulares."

- Ya probamos que $LR^{\Sigma} \equiv AF^{\Sigma}$ y $GR^{\Sigma} \equiv GRPU^{\Sigma}$.
- Ahora, probaremos que $GRPU^{\Sigma} \equiv AF^{\Sigma}$.
- Y por transitividad, tendremos que $LR^{\Sigma} \equiv GR^{\Sigma}$.

Teorema ($GRPU^{\Sigma} \equiv AF^{\Sigma}$)

1. *(ida) Si* $G \in GRPU^{\Sigma}$, *entonces* $\exists M \in AF^{\Sigma}$ *tal que* $L(G) = L(M)$.
2. *(vuelta) Si* $M \in AF^{\Sigma}$, *entonces* $\exists G \in GRPU^{\Sigma}$ *tal que* $L(G) = L(M)$.

Demo 1. Sea $G = (V, \Sigma, S, P)$, entonces podemos tomar $M = (\Sigma, Q, q_0, F, \Delta)$ donde:

$Q = V$
$q_0 = S$
$F = \{A \in V : A \to \epsilon \in P\}$
$\Delta(A, a) = \{B \in V : A \to aB \in P\}$
$\Delta(A, \epsilon) = \{B \in V : A \to B \in P\}$

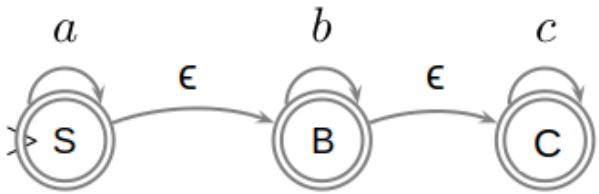
Claramente, M es un AFNϵ.
Luego, $\alpha \in L(G)$ sii $\alpha \in L(M)$ $\therefore L(G) = L(M)$.

Por ejemplo, sea G la siguiente GRPU:

$$\begin{aligned} G : S &\to aS|B|\epsilon \\ B &\to bB|C|\epsilon \\ C &\to cC|\epsilon \end{aligned}$$

con $L(G) = a^*b^*c^*$.

Su AFNϵ equivalente M sería:



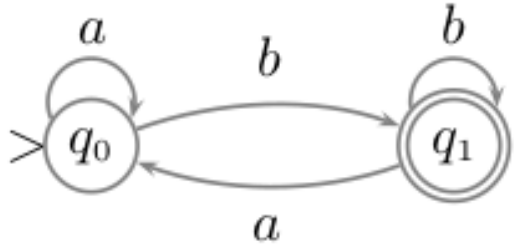
Y claramente, $L(M) = a^*b^*c^*$.

Demo 2. Sea $M = (\Sigma, Q, q_0, F, \Delta)$ AFNϵ, entonces podemos tomar $G = (V, \Sigma, S, P)$ donde:

$V = Q$
$S = q_0$
$P = \{q \to aq' : q' \in \Delta(q, a)\} \cup$
$\{q \to q' : q' \in \Delta(q, \epsilon)\} \cup$
$\{q \to \epsilon : q \in F\}$

Claramente, G es una GRPU.
Luego, $\alpha \in L(M)$ sii $\alpha \in L(G)$ $\therefore L(G) = L(M)$.

Por ejemplo, sea M el siguiente AF:



con $L(M) = (a + b)^* b$.

Su GRPU G equivalente con variable inicial q_0 sería:

$$\begin{aligned} G : q_0 &\to aq_0 | bq_1 \\ q_1 &\to bq_1 | aq_0 | \epsilon \end{aligned}$$

Y claramente, $L(G) = (a + b)^* b$.

Resumen

- Presentamos el modelo gramatical propuesto por Chomsky, el cual define cada una de las clases de lenguajes en su jerarquía.
- Luego, definimos la noción de gramática regular, a partir de una fuerte restricción en la forma de las producciones de la gramática.
- Probamos que un lenguaje es regular si y solo si es generado por una gramática regular.
- Por lo tanto, las gramáticas regulares son también, al igual que los AF, un modelo computacional caracterizador de los lenguajes regulares.
- Tenemos una nueva herramienta para probar regularidad de un lenguaje simplemente dando una GR que lo genera.

Lenguajes No Regulares

Introducción a la Logica y la Computación (3era Parte)

Docentes: Badano, Bustos, Costamagna, Tellechea, Zigaran

Año 2024

Lenguajes No-Regulares

- Hasta aquí, todos los lenguajes vistos resultaron ser lenguajes regulares.
- Por lo tanto, surge la pregunta sobre si verdaderamente existen lenguajes que no lo sean.
- La respuesta es si, claro que existen y hay infinidad de ellos.

Por ejemplo, el lenguaje de todas las cadenas que son secuencia de a's seguido de una secuencia de b's de igual longitud no es regular:

$$L_1 = \{a^m b^m : m \geq 0\} \notin LR^{\Sigma}.$$

Una intuición posible detrás de este resultado, es que no puede existir un AF que compute L_1, pues los AF son máquinas puramente locales (sin memoria). No guardan ningún registro de lo que ya computaron de la cadena. Y claramente, para poder computar L_1 necesitamos de algún mecanismo para saber cuantas a's procesamos previamente para poder compararla más tarde con la cantidad de b's que se procesarán.

Más lenguajes No-Regulares

El lenguaje de las cadenas capicuas en el alfabeto no es regular:

$$L_2 = \{\alpha \in \Sigma^* : \alpha = \alpha^R\} \notin LR^{\Sigma}.$$

El lenguaje de todas las reflexiones en el alfabeto no es regular:

$$L_3 = \{\alpha\alpha : \alpha \in \Sigma^*\} \notin LR^{\Sigma}.$$

Ninguno de estos tres lenguajes puede tener un AF que lo acepte.

L_1 y L_2 son lenguajes independientes de contexto, mientras que L_3 es sensible al contexto.

Técnica Formal para Probar No-Regularidad

- Más alla de la intuición, necesitamos una herramienta para probar con **rigurosidad matemática** que dicho autómoma no existe, es decir, una técnica que pruebe formalmente que un lenguaje no es regular.
- Para ello, utilizaremos el **"Pumping Lema"** que es una propiedad que todo lenguaje regular cumple.
- Por lo tanto, si probamos que un lenguaje no satisface la propiedad, entonces ese lenguaje no puede ser regular.

Pumping Lema (o Lema de Bombeo)

La propiedad dice que toda cadena de un lenguaje regular tiene una subcadena que puede repetirse una cantidad arbitraria de veces (bombeo) y esta nueva cadena "bombeada" pertenece al lenguaje.

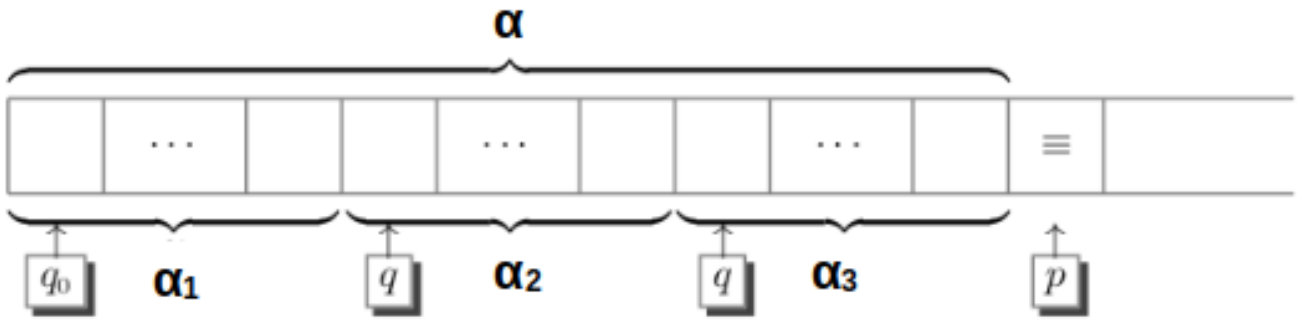
Lema (Pumping Lema)

Para todo $L \in LR^{\Sigma}$, *existe* $n \in \mathbb{N}$, *llamada constante de bombeo de* L, *tal que, si* $\alpha \in L$ *con* $|\alpha| \geq n$, *entonces* α *se puede descomponer como* $\alpha = \alpha_1\alpha_2\alpha_3$ *donde:*

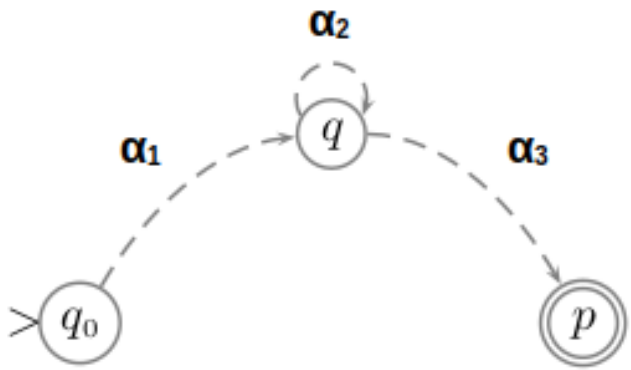
1) $|\alpha_1\alpha_2| \leq n$
2) $\alpha_2 \neq \epsilon$
3) $(\alpha_1\alpha_2^i\alpha_3) \in L$, *para todo* $i \in \mathbb{N}$

Demo. Si $L \in LR^{\Sigma}$, entonces existe un AFD $M = (\Sigma, Q, q_0, F, \delta)$ tal que $L(M) = L$ y sea $n = |Q|$.
Si $\alpha \in L$ y $|\alpha| \geq n$, entonces al procesar la cadena α existe al menos un estado $q \in Q$ que es visitado dos veces (o sea, se repite).

Por lo tanto, notar que α se puede descomponer de la siguiente manera:



Luego, la subcadena α_2 puede ser repetida o bombeada una cantidad cualquiera de veces, y aún asi, el procesamiento de la cadena bombeada arribará al estado $p \in F$, por lo tanto, pertenece al lenguaje L:



$(\alpha_1 \alpha_2^i \alpha_3) \in L$, para todo $i \geq 0$

Pasos para Probar No-Regularidad

Para demostrar formalmente que un lenguaje L no es regular, utilizando el Pumping Lema, procedemos siempre de la siguiente manera:

1) Suponer que $L \in LR^{\Sigma}$ y denotar con n a la cte de bombeo de L.

2) Tomar una $\alpha \in L$, con $|\alpha| \geq n$, arbitraria pero "conveniente" que esté definida en términos de n.

3) Descomponer α en subcadenas $\alpha_1, \alpha_2, \alpha_3$ como indica el Pumping Lema.

4) Hallar un $i \geq 0$ tal que $(\alpha_1 \alpha_2^i \alpha_3) \notin L$ (Absurdo!!!)

El absurdo proviene de suponer en (1) que $L \in LR^{\Sigma}$ $\therefore$ $L \notin LR^{\Sigma}$.

Aplicando el Pumping Lema

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

Veamos que $L_1 = \{a^m b^m : m \geq 0\} \notin LR^{\Sigma}$.

Sup. $L_1 \in LR^{\Sigma}$ y sea n la cte de bombeo de L_1.

Tomamos $\alpha = a^n b^n \in L_1$, luego $|\alpha| = 2n \geq n$.

Por Pumping Lema, $\alpha = \alpha_1\alpha_2\alpha_3$ donde:

$\alpha_1 = a^r$ con $r \geq 0$
$\alpha_2 = a^s$ con $s \geq 1$
$\alpha_3 = a^{n-(s+r)} b^n$

Para $i = 0$, tenemos que:

$$\begin{aligned}\alpha_1\alpha_2^0\alpha_1 &= a^r(a^s)^0 a^{n-(s+r)} b^n \\ &= a^r \epsilon a^{n-s-r} b^n \\ &= a^{n-s} b^n \notin L_1, \text{ pues } s \geq 1 \therefore \text{Absurdo.}\end{aligned}$$

$\therefore L_1 \notin LR^{\Sigma}$.

Veamos que $L_2 = \{\alpha \in \Sigma^* : \alpha = \alpha^R\} \notin LR^{\Sigma}$, es decir, el lenguaje de todas las cadenas capicuas del alfabeto no es regular.

Sup. $L_2 \in LR^{\Sigma}$ y sea n la cte de bombeo de L_2.

Tomamos $\alpha = a^n b a^n \in L_2$, luego $|\alpha| = 2n + 1 \geq n$.

Por Pumping Lema, $\alpha = \alpha_1\alpha_2\alpha_3$ donde:

$\alpha_1 = a^r$ con $r \geq 0$
$\alpha_2 = a^s$ con $s \geq 1$
$\alpha_3 = a^{n-(s+r)} b a^n$

Para $i = 2$, tenemos que:

$$\begin{aligned}\alpha_1\alpha_2^2\alpha_3 &= a^r (a^s)^2 a^{n-(s+r)} b a^n \\ &= a^r a^{2s} a^{n-s-r} b a^n \\ &= a^{n+s} b a^n \notin L_1, \text{ pues } s \geq 1 \therefore \text{ Absurdo.}\end{aligned}$$

$\therefore L_2 \notin LR^{\Sigma}$.

Resumen

- Hay lenguajes que no son regulares tales como $L_1 = \{a^m b^m : m \geq 0\}$ y $L_2 = \{\alpha \in \Sigma^* : \alpha = \alpha^R\}$.
- El Pumping Lema y razonamiento por el absurdo conforman una técnica para probar formalmente que un lenguaje no es regular.
- L_1 y L_2 son lenguajes independientes de contexto y esto se prueba mediante autómatas a pila y/o gramáticas independientes de contexto (modelos que serán estudiados en 4to año).