Pontificia Universidad Católica de Chile

Departamento de Ciencia de la Computación

Apunte IIC2223

Teoría de Autómatas y Lenguajes Formales

Autor
Cristóbal Rojas

En base a apuntes de
Prof. Cristian Riveros

3 de diciembre de 2022

Índice

1. Lenguajes regulares

1.1. Palabras y autómatas

1.1.1. Palabras

Definiciones. Consideremos que:

♦ Un **alfabeto** Σ es con conjunto finito.

♦ Un elemento de Σ lo llamaremos una **letra** o **símbolo**.

♦ Una **palabra** o **string** sobre Σ es una secuencia finita de letras en Σ.

Ejemplo 1.1

- $\Sigma = \{a, b, c\}$
- Palabras sobre Σ:
$$aaaaabb,\ bcaabab,\ a,\ bbbbbb,\ \cdots$$

♦ El largo $|w|$ de una palabra w es el número de letras.

$$\boxed{|w| \overset{\text{def}}{\equiv} \#\text{ de letras en } w}$$

♦ Denotaremos ϵ como la **palabra sin símbolos** de largo 0.

$$\boxed{|\epsilon| \overset{\text{def}}{\equiv} 0}$$

♦ Denotaremos por Σ^* como el **conjunto de todas las palabras** sobre Σ.

Ejemplo 1.2

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

- $|00011001| = 8$
- Σ^* = todas las palabras posibles formadas por 0s y 1s.

Definición. Dados dos palabras $u, v \in \Sigma^*$ tal que $u = a_1 \dots a_n$ y $v = b_1 \dots v_m$:

$$\boxed{u \cdot v \overset{\text{def}}{\equiv} a_1 \dots a_n b_1 \cdots b_m}$$

Decimos que $u \cdot v$ es la palabra "u **concatenada** con v".

Ejemplo 1.3

Para $\Sigma = \{0, 1, 2, \dots, 9\}$:

♦ $0123 \cdot 9938 = 01239938$ y $3493 \cdot \epsilon = 3493$

Propiedades de concatenación. La concatenación cumple:

- **Asociatividad:** $(u \cdot v) \cdot w = u \cdot (v \cdot w)$
- **Largo:** $|u \cdot v| = |u| + |v|$
- ¿Cumple conmutatividad? No. Por ejemplo, si $u = ab$ y $v = bb$, entonces $u \cdot v = abbb \neq bbab = v \cdot u$.

Definición. Sea Σ un alfabeto y $L \subseteq \Sigma^*$. Decimos que L es un **lenguaje** sobre el alfabeto Σ.

Ejemplo 1.4

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

- $L_0 = \{\epsilon, a, aa, b, aa\}$
- $L_1 = \{\epsilon, b, bb, bbb, bbbb, \ldots\}$
- $L_2 = \{w \mid \exists u \in L_1, w = a \cdot u\}$
- $L_3 = \{w \mid \exists u, v \in \Sigma^*, w = u \cdot abba \cdot v\}$
- $L_4 = \{w \mid \exists u \in \Sigma^*, w = u \cdot u\}$

Un **lenguaje** puede ser visto como una **propiedad** de palabras.

Convenciones. Durante todo este texto:

- Para **letras** se usarán los símbolos: $a, b, c, d, e, \ldots$
- Para **palabras** se usarán los símbolos: $w, u, v, x, y, z, \ldots$
- Para **alfabetos** se usarán los símbolos: $\Sigma, \Gamma, \ldots$
- Para **lenguajes** se usarán los símbolos: $L, M, N, \ldots$
- Para **números** se usarán los símbolos: $i, j, k, l, m, n, \ldots$

1.1.2. Autómatas

Una autómata **finito** es:

- Un modelo de computación sencillo, basado en una cantidad **finita** de memoria.
- Procesa cada **palabra** de principio a fin en **una sola pasada**.
- Al terminar, el autómata decide si **acepta** o **rechaza** el input.

Usaremos los autómatas finitos para definir **lenguajes**.

Definición. Un autómata finito determinista (DFA) es una tupla:

$$\boxed{\mathcal{A} = (Q, \Sigma, \delta, q_0, F)}$$

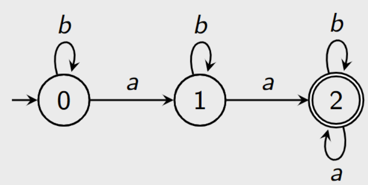
- Q es un conjunto finito de **estados**.
- Σ es el alfabeto del **input**.
- $\delta : Q \times \Sigma \to Q$ es la función de **transición**.
- $q_0 \in Q$ es el **estado inicial**.
- $F \subseteq Q$ es el conjunto de **estados finales** (o aceptación).

Ejemplo 1.5

- $Q = \{0, 1, 2\}$
- $\Sigma = \{a, b\}$
- $\delta : Q \times \Sigma \to Q$ se define como:

$$\begin{aligned} \delta(0, a) &= 1 \\ \delta(1, a) &= 2 \\ \delta(2, a) &= 2 \\ \delta(q, b) &= q \quad \forall q \in \{0, 1, 2\} \end{aligned}$$

- $q_0 = 0$
- $F = \{2\}$



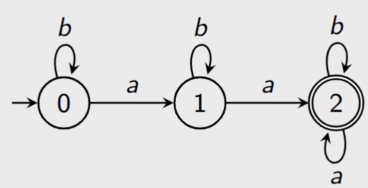
Ejecución. Sea $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un DFA y $w = a_1 a_2 \ldots a_n \in \Sigma^*$ un input. Una **ejecución** (o *run*) ρ de $\mathcal{A}$ sobre w es una secuencia:

$$\boxed{\rho : \quad p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \ldots \xrightarrow{a_n} p_n}$$

donde $p_0 = q_0$ y para todo $i \in \{0, 1, \ldots, n-1\}$ se cumple que $\delta(p_i, a_{i+1}) = p_{i+1}$.

Una ejecución ρ de $\mathcal{A}$ sobre w es de **aceptación** si $p_n \in F$.

Ejemplo 1.6



- ¿Cuál es la ejecución de $\mathcal{A}$ sobre *bbab*?

  $\rho : 0 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{b} 1$. La ejecución **no** es de aceptación ya que no termina en un estado final.

- ¿Cuál es la ejecución de $\mathcal{A}$ sobre *abab*?

  $\rho : 0 \xrightarrow{a} 1 \xrightarrow{b} 1 \xrightarrow{a} 2 \xrightarrow{b} 2$. La ejecución **si** es de aceptación ya que termina en un estado final.

Aceptación. Sea $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un DFA y $w \in \Sigma^*$. Decimos que $\mathcal{A}$ **acepta** w si la ejecución de $\mathcal{A}$ sobre w es de aceptación. Al contrario, decimos que $\mathcal{A}$ **rechaza** w si la ejecución de $\mathcal{A}$ sobre w NO es de aceptación.

El **lenguaje aceptado** por $\mathcal{A}$ se define como:

$$\boxed{\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ acepta } w\}}$$

Un lenguaje $L \subseteq \Sigma^*$ se dice **regular** si, y sólo si, **existe** un autómata finito determinista $\mathcal{A}$ tal que

$$\boxed{L = \mathcal{L}(\mathcal{A})}$$

1.2. Construcciones de autómatas

Veremos una definición alternativa al autómata visto en la sección anterior.

1.2.1. Autómatas con función parcial de transición

Definición. Un autómata finito determinista con **función parcial de transición** (DFAp) es una tupla:

$$\boxed{\mathcal{A} = (Q, \Sigma, \gamma, q_0, F)}$$

- Q es un conjunto finito de estados.
- Σ es el alfabeto del input.
- $\gamma : Q \times \Sigma \rightharpoonup Q$ es una **función parcial de transición**.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales (o aceptación).

Ejecución. Sea $w = a_1 a_2 \dots a_n \in \Sigma^*$. De igual manera que un DFA, una **ejecución** (o *run*) ρ de $\mathcal{A}$ sobre w es una secuencia:

$$\boxed{\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots \xrightarrow{a_n} p_n}$$

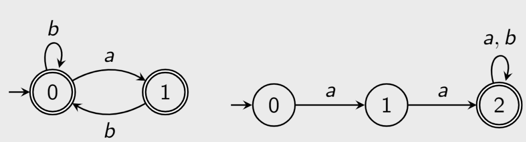
donde $p_0 = q_0$ y para todo $i \in \{0, \dots, n-1\}$ **está definido** $\gamma(p_i, a_{i+1}) = p_{i+1}$.

Una ejecución ρ de $\mathcal{A}$ sobre w es de **aceptación** si $p_n \in F$. Notemos que ahora una palabra puede NO tener una ejecución.

Aceptación. Sea $\mathcal{A}$ un DFAp y $w \in \Sigma^*$. Decimos que $\mathcal{A}$ **acepta** w si **existe una ejecución** de $\mathcal{A}$ sobre w que es de aceptación. También, el **lenguaje aceptado** por $\mathcal{A}$ se define como:

$$\boxed{\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ acepta } w\}}$$

Ejemplo 1.7



¿DFA $\not\equiv$ DFAp? Establezcamos una proposición. Para todo autómata $\mathcal{A}$ con función parcial de transición, existe un autómata $\mathcal{A}'$ (con función total de transición) tal que:

$$\boxed{\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')}$$

En otras palabras, DFA $\equiv$ DFAp.

Demostración. Sea $\mathcal{A} = (Q, \Sigma, \gamma, q_0, F)$ un autómata con función parcial de transición. Sea q_s un **nuevo estado** tal que $q_s \notin Q$. Contruimos el DFA $\mathcal{A}' = (Q \cup \{q_s\}, \Sigma, \delta', q_0, F)$ tal que:

$$\delta'(p, a) = \begin{cases} \gamma(p, a) & \text{si } p \neq q_s \text{ y } (p, a) \in \text{dom}(\gamma) \\ q_s & \text{si no} \end{cases}$$

para todo $p \in Q \cup \{q_s\}$ y $a \in \Sigma$. Queremos demostrar que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Dem. $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$. Sea $w = a_1 \dots a_n \in \mathcal{L}(\mathcal{A})$. Entonces, existe una ejecución de aceptación ρ de $\mathcal{A}$ sobre w:

$$\rho : p_0 \overset{a_1}{\to} p_1 \overset{a_2}{\to} p_2 \dots \overset{a_n}{\to} p_n$$

donde $p_0 = q_0$, para todo $i \in \{0, \dots, n-1\}$ está definido $\gamma(p_i, a_{i+1}) = p_{i+1}$ y $p_n \in F$. Como $\delta'(p_i, a_{i+1}) = \gamma(p_i, a_{i+1})$ para todo $i \in \{0, \dots, n-1\}$ (por la definición de δ'), entonces ρ es también una ejecución de aceptación de $\mathcal{A}'$ sobre w. Por lo tanto, $w \in \mathcal{L}(\mathcal{A}')$.

Dem. $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$. Sea $w = a_1 \dots a_n \in \mathcal{L}(\mathcal{A}')$. Entonces, existe una ejecución de aceptación ρ de $\mathcal{A}'$ sobre w:

$$\rho : p_0 \overset{a_1}{\to} p_1 \overset{a_2}{\to} p_2 \dots \overset{a_n}{\to} p_n$$

donde $p_0 = q_0$, para todo $i \in \{0, \dots, n-1\}$ tenemos $\gamma'(p_i, a_{i+1}) = p_{i+1}$ y $p_n \in F$. Demostraremos que $p_i \neq q_s$ para todo $i \in \{0, \dots, n\}$. Por **contradicción**, suponga que existe i tal que $p_i = q_s$. entonces, tenemos que $p_{i+1} = q_s$. Por **inducción**, podemos demostrar que $p_j = q_s$ para todo $j \geq i$, y así, podemos concluir que $p_n = q_s$, llevándonos a una contradicción. Como $p_i \neq q_s$ para todo $i \in \{0, \dots, n\}$, tenemos que:

$$\delta'(p_i, a_{i+1}) = \gamma(p_i, a_{i+1}) \quad \forall i \in \{0, 1, \dots, n-1\}$$

y entonces ρ es una **ejecución de aceptación** de $\mathcal{A}$ sobre w. Por lo tanto, concluimos que $w \in \mathcal{L}(\mathcal{A})$. ■

Advertencia. Desde ahora, se utilizaran autómatas con funciones **totales** de transición, pero sin pérdida de generalidad, en algunos ejemplos habrán autómatas con funciones **parciales** de transición por simplicidad.

1.2.2. Operaciones de conjuntos

Definiciones. Dado dos lenguajes $L, L' \subseteq \Sigma^*$ se define:

$$\boxed{\begin{aligned} L^C &= \{w \in \Sigma^* \mid w \notin L\} \\ L \cap L' &= \{w \in \Sigma^* \mid w \in L \land w \in L'\} \\ L \cup L' &= \{w \in \Sigma^* \mid w \in L \lor w \in L'\} \end{aligned}}$$

Dado dos autómatas $\mathcal{A}$ y $\mathcal{A}'$:

1. ¿Existe un autómata $\mathcal{B}$ tal que $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A})^C$?
2. ¿Existe un autómata $\mathcal{B}$ tal que $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$?
3. ¿Existe un autómata $\mathcal{B}$ tal que $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{A}')$?

Construcción de $\mathcal{L}(\mathcal{A})^C$. Dado un autómata $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, definimos el autómata:

$$\boxed{\mathcal{A}^C = (Q, \Sigma, \delta, q_0, Q \backslash F)}$$

Teorema 1

Para todo autómata $\mathcal{A}$, se tiene que $\mathcal{L}(\mathcal{A})^C = \mathcal{L}(\mathcal{A}^C)$.

Producto de autómatas. Suponga que:

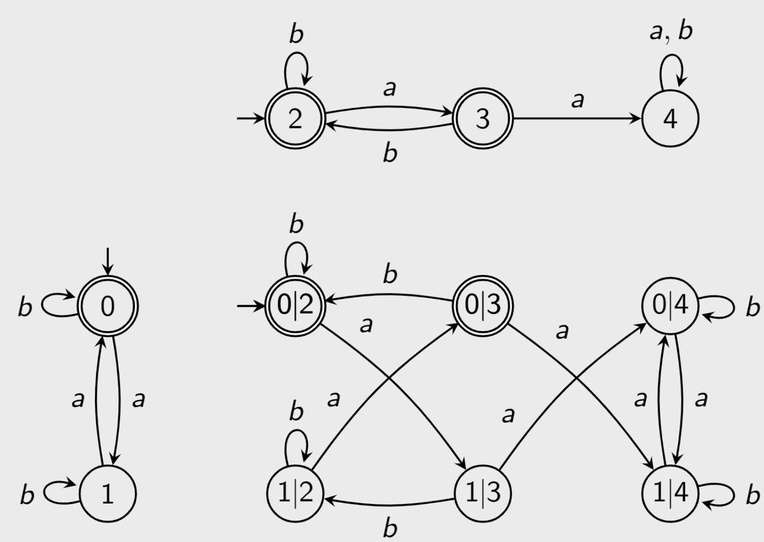
$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$$
$$\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$$

y considere una palabra $w \in \Sigma^*$. ¿Cómo ejecutamos ambos autómatas sobre w al **mismo tiempo**? La idea es ejecutar $\mathcal{A}$ y $\mathcal{A}'$ **en paralelo**. Así, definimos el **producto** entre $\mathcal{A}$ y $\mathcal{A}'$ como el autómata $\mathcal{A} \times \mathcal{A}' = (Q^\times, \Sigma, \delta^\times, q_0^\times, F^\times)$ tal que:

- $Q^\times = Q \times Q' = \{(q, q') \mid q \in Q \wedge q' \in Q'\}$
- $\delta^\times((q, q'), a) = (\delta(q, a), \delta'(q', a))$
- $q_0^\times = (q_0, q_0')$
- $F^\times = F \times F'$

Ejemplo 1.8

Todas las palabras sobre $\{a, b\}$ con una cantidad par de a-letras tal que no hay dos a-letras seguidas.



Teorema 2

Para todo par de autómatas $\mathcal{A}$ y $\mathcal{A}'$ se tiene que

$$\mathcal{L}(\mathcal{A} \times \mathcal{A}') = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$$

Demostración teorema 2. Solo se demostrará que $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A} \times \mathcal{A}')$, la otra dirección queda propuesta para el lector.

Sea $w = a_1 \dots a_n \in \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$. Entonces $w \in \mathcal{L}(\mathcal{A})$ y $w \in \mathcal{L}(\mathcal{A}')$. Existen ejecuciones de aceptación ρ y ρ' de $\mathcal{A}$ y $\mathcal{A}'$ sobre w, respectivamente:

$$\rho : p_0 \overset{a_1}{\to} p_1 \overset{a_2}{\to} \dots \overset{a_n}{\to} p_n \quad \rho' : p_0' \overset{a_1}{\to} p_1' \overset{a_2}{\to} \dots \overset{a_n}{\to} p_n'$$

- $p_0 = q_0$ y $p_0' = q_0'$.
- $\delta(p_{i-1}, a_i) = p_i$ y $\delta'(p_{i-1}', a_i) = p_i$ para todo $i \in \{1, \dots, n\}$.
- $p_n \in F$ y $p_n' \in F'$.

Por definición, tenemos que: $\rho^{\times} : (p_0, p_0') \overset{a_1}{\to} (p_1, p_1') \overset{a_2}{\to} \dots \overset{a_n}{\to} (p_n, p_n')$

- $(p_0, p_0') = (q_0, q_0')$.
- $(p_i, p_i') = \left(\delta\left(p_{i-1}, a_i\right), \delta'\left(p_{i-1}', a_i\right)\right) = \delta^{\times}\left(\left(p_{i-1}, p_{i-1}'\right), a_i\right) \forall i \in \{1, \dots, n\}$.
- $(p_n, p_n') \in F \times F'$.

Por lo tanto, $\rho^{\times}$ es una ejecución de $\mathcal{A} \times \mathcal{A}'$ sobre w y $w \in \mathcal{L}(\mathcal{A} \times \mathcal{A}')$. ■

Unión de autómatas. Sabemos que

$$\mathcal{L}(\mathcal{A}) \cup \mathcal{L}\left(\mathcal{A}'\right) = \left(\mathcal{L}(\mathcal{A})^C \cap \mathcal{L}\left(\mathcal{A}'\right)^C\right)^C$$

Para calcular el autómata que acepta el lenguaje $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{A}')$:

1. Complementamos $\mathcal{A}$ y $\mathcal{A}'$.
2. Intersectamos $\mathcal{A}^C$ y $(\mathcal{A}')^C$.
3. Complementamos $\mathcal{A}^C \times (\mathcal{A}')^C$.

1.3. No-determinismo

"Indeterminism is the concept that events (certain events, or events of certain types) are not caused deterministically (cf. causality) by prior events. It is the opposite of ***determinism*** *and related to chance. It is highly relevant to the philosophical problem of* ***free will****."* - Wikipedia.

1.3.1. Definición de un NFA

Definición. Un autómata finito **no-determinista** (NFA) es una estructura:

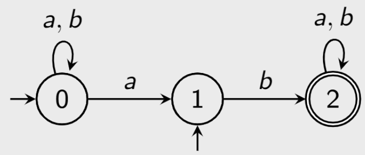
$$\boxed{\mathcal{A} = (Q, \Sigma, \Delta, I, F)}$$

- Q es un conjunto finito de estados.
- Σ es el alfabeto del input.
- $F \subseteq Q$ es el conjunto de estados finales (o aceptación).
- $\Delta \subseteq Q \times \Sigma \times Q$ es la **relación de transición**.
- $I \subseteq Q$ es un **conjunto de estados iniciales**.

Ejemplo 1.9

- $Q = \{0,1,2\}$, $\Sigma = \{a,b\}$, $I = \{0,1\}$, $F = \{2\}$
- $\Delta \subseteq Q \times \Sigma \times Q$ se define como:

$$\begin{array}{rcl} (0,a,0) & \in & \Delta \\ (0,a,1) & \in & \Delta \\ (0,b,0) & \in & \Delta \\ (1,b,2) & \in & \Delta \\ (2,a,2) & \in & \Delta \\ (2,b,2) & \in & \Delta \end{array}$$



Ejecución. Sea $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ un NFA y $w = a_1 a_2 \dots a_n \in \Sigma^*$ el input. Una **ejecución** (o *run*) ρ de $\mathcal{A}$ sobre w es una secuencia:

$$\boxed{\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n}$$

donde $p_0 \in I$ y para todo $i \in \{0, \dots, n-1\}$, se tiene que $(p_i, a_{i+1}, p_{i+1}) \in \Delta$.

Una ejecución ρ de $\mathcal{A}$ sobre w es de **aceptación** si $p_n \in F$.

Aceptación. Decimos que $\mathcal{A}$ **acepta** w si **existe** una ejecución de $\mathcal{A}$ sobre w que es de aceptación. Por otro lado, decimos que $\mathcal{A}$ **rechaza** si **todas** las ejecuciones de $\mathcal{A}$ sobre w NO son de aceptación. Además, el **lenguaje aceptado** por $\mathcal{A}$ se define como:

$$\boxed{\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ acepta } w\}}$$

Interpretación. Las siguientes interpretaciones pueden ayudar a entender mejor un NFA:

1. $\Delta \subseteq Q \times \Sigma \times Q$ es la **relación de transición**.
   "$(q, a, p) \in \Delta$ entonces existe una transición desde q a p al leer a".
2. $I \subseteq Q$ es un **conjunto de estados iniciales**.
   "$p \in I$ entonces p es un posible estado inicial del autómata"

1'. $\Delta : Q \times \Sigma \to 2^Q$ es una **función de transición**.
   "$q \in \Delta(p, a)$ entonces q es un posible estado que puedo llegar desde p al leer a".
   Esta interpretación es más común encontrarla en libros sobre teoría de autómatas.

Además, el **no-determinismo** puede ser visto como:

1. Paralelización infinita, es decir, cada ejecución es un *thread* distinto.
2. "Guessing and Verifying" (adivinar y verificar).

El no-determinismo NO debe ser visto como:

- Explicitamente como el **indeterminismo** o "libre albedrío". Para un input, un NFA siempre produce el mismo resultado.
- Comportamiento **aleatorio** del autómata.

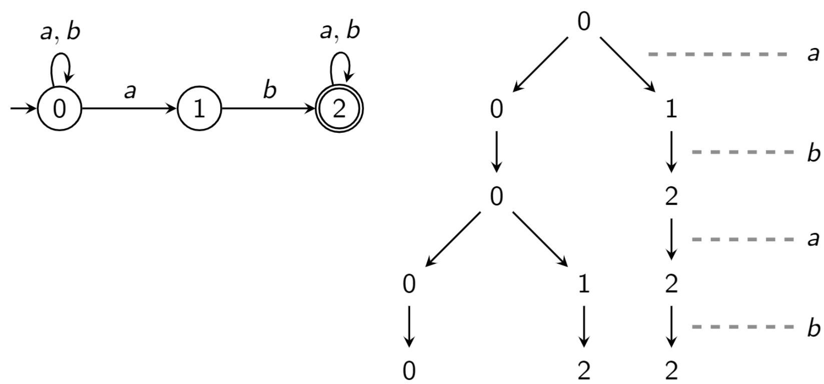


Figura 1: Interpretación del no-determinismo

1.3.2. Comparación con DFA

A continuación, veremos que autómata finito determinista (DFA) puede almacenar **todas** las ejecuciones de un NFA. A este proceso se le conoce como **determinización**.

Teorema 3

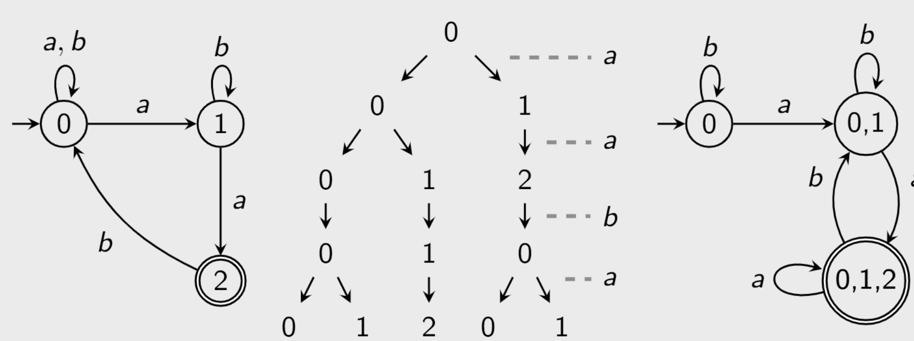
Para todo autómata finito no-determinista $\mathcal{A}$, existe un autómata determinista $\mathcal{A}'$ tal que

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$$

En otras palabras, DFA $\equiv$ NFA.

Idea. Primero, pensemos en la idea de determinización: *"almacenar en el autómata determinista todos los estados actuales de las ejecuciones en curso (sin repetidos)"*.

Ejemplo 1.10



Formalización. Para un autómata no-determinista $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, definimos el autómata determinista (**determinización** de $\mathcal{A}$):

$$\boxed{\mathcal{A}^{\text{det}} = (2^Q, \Sigma, \delta^{\text{det}}, q_0^{\text{det}}, F^{\text{det}})}$$

- $2^Q = \{S \mid S \subseteq Q\}$ es el conjunto potencia de Q.
- $q_0^{\text{det}} = I$.
- $\delta^{\text{det}} : 2^Q \times \Sigma \to 2^Q$ tal que:

$$\delta^{\text{det}}(S, a) = \{q \in Q \mid \exists p \in S.\ (p, a, q) \in \Delta\}$$

- $F^{\text{det}} = \{S \in 2^Q \mid S \cap F \neq \varnothing\}$, es decir, todos los conjuntos que tengan al menos un estado final.

Demostración teorema 3. La determinización puede verse como un **subset construction**. Partamos con $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}^{\text{det}})$.

Sea $w = a_1 \dots a_n \in \mathcal{L}(\mathcal{A})$. Existe una ejecución ρ de $\mathcal{A}$ sobre w:

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n$$

donde $p_0 = I$, $(p_i, a_{i+1}, p_{i+1}) \in \Delta$ para todo $i \in \{0, \dots, n-1\}$ y $p_n \in F$.

Como $\mathcal{A}^{\text{det}}$ es determinista, entonces existe una ejecución ρ' de $\mathcal{A}^{\text{det}}$ sobre w:

$$\rho' : S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} S_n$$

donde $S_0 = I$ y $\delta^{\text{det}}(S_i, a_{i+1}) = S_{i+1}$ para todo $i \in \{0, \dots, n-1\}$. Luego, queremos demostrar que $p_i \in S_i$ para todo $i \in \{0, \dots, n-1\}$.

Por **inducción** sobre i, tenemos que:

- **Caso base:** $p_0 \in S_0$ por definición de $\mathcal{A}^{\text{det}}$.
- **Inducción:** Suponemos que $p_i \in S_i$ y demostramos para $i+1$. Como sabemos que:
  - $\delta^{\text{det}}(S_i, a_{i+1}) = S_{i+1} = \{q \in Q \mid \exists p \in S_i.\ (p, a, q) \in \Delta\}$ y
  - $(p_i, a_{i+1}, p_{i+1}) \in \Delta$

  Entonces $p_{i+1} \in S_{i+1}$, ya que, si estamos en p_i leyendo a_{i+1}, la transición nos dice que pasaremos al estado p_{i+1} que pertenece a S_{i+1} por la hipótesis de inducción.

Luego, como $p_n \in S_n$, entonces $S_n \cap F \neq \varnothing$ y así $S_n \in F^{\text{det}}$. Por lo tanto, $w \in \mathcal{L}(\mathcal{A}^{\text{det}})$.

Ahora, demostramos la otra dirección: $\mathcal{L}(\mathcal{A}^{\text{det}}) \subseteq \mathcal{L}(\mathcal{A})$.

Sea $w = a_1 \dots a_n \in \mathcal{L}(\mathcal{A}^{\text{det}})$. Existe una ejecución ρ de $\mathcal{A}^{\text{det}}$ sobre w:

$$\rho : S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} S_n$$

donde $S_0 = I$, $\delta^{\text{det}}(S_i, a_{i+1}) = S_{i+1}$ para todo $i \in \{0, \dots, n-1\}$ y $S_n \in F^{\text{det}}$, con $S_n \cap F \neq \varnothing$. Buscamos demostrar entonces que para todo $i \leq n$ y para todo $p \in S_i$, existe una ejecución:

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_i} p_i = p$$

tal que:

1. $p_0 \in I$.
2. $(p_j, a_{j+1}, p_{j+1}) \in \Delta$ para todo $j \in \{0, \dots, i-1\}$.

Por **inducción** sobre i, tenemos que:

- **Caso base:** Si $p \in S_0 = I$, entonces la ejecución $\rho : p$ cumple 1. y 2.
- **Inducción:** Supongamos que se cumple para todo $p \in S_i$. Sea $q \in S_{i+1}$. Como $\delta^{\text{det}}(S_i, a_{i+1}) = S_{i+1} = \{q \in Q \mid \exists p \in S_i.\ (p, a, q) \in \Delta\}$ y $q \in S_{i+1}$, entonces existe $p \in S_i$ tal que $(p, a_{i+1}, q) \in \Delta$.

Por hipótesis de inducción, existe $\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_i} p_i = p$ que satisface 1. y 2.

Por lo tanto, $\rho' : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_i} p_i \xrightarrow{a_{i+1}} q$ también satisface 1. y 2.

Como lo anterior queda demostrado y como $S_n \cap F \neq \varnothing$, para $p \in S_n \cap F$ existe una ejecución de aceptación de $\mathcal{A}$ sobre w. Por lo tanto, $w \in \mathcal{L}(\mathcal{A})$. ■

1.4. Expresiones regulares

Definición. R es una **expresión regular** sobre Σ si R es igual a:

1. a, para alguna letra $a \in \Sigma$.
2. ϵ
3. $\varnothing$
4. $(R_1 + R_2)$, donde R_1 y R_2 son expresiones regulares.
5. $(R_1 \cdot R_2)$, donde R_1 y R_2 son expresiones regulares.
6. (R_1^*), donde R_1 es una expresión regular. Esta expresión se conoce como **clausura de Kleene**.

Denotaremos como ExpReg el conjunto de **todas las expresiones regulares** sobre Σ.

Ejemplo 1.11

Las siguientes son expresiones regulares sobre $\Sigma = \{a, b\}$:

- $(a + b)$
- $((a \cdot b) \cdot c)$
- (a^*)
- $(b \cdot (a^*))$
- $((a + b)^*)$
- $((a \cdot ((b \cdot a)^*)) + \epsilon)$
- $((a \cdot ((b \cdot a)^*)) + \varnothing)$

Para reducir la cantidad de paréntesis, se define el **orden de precendencia**:

1. estrella $(\cdot)^*$
2. concanetación $\cdot$
3. unión $+$

Semántica. Para una expresión regular R cualquiera, se define el lenguaje $\mathcal{L}(R) \subseteq \Sigma^*$ **inductivamente** como:

1. $\mathcal{L}(a) = \{a\}$, para toda letra $a \in \Sigma$.
2. $\mathcal{L}(\epsilon) = \{e\}$.
3. $\mathcal{L}(\varnothing) = \varnothing$.
4. $\mathcal{L}(R_1 + R_2) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$, donde R_1 y R_2 son expresiones regulares.
5. $\mathcal{L}(R_1 \cdot R_2) = \mathcal{L}(R_1) \cdot \mathcal{L}(R_2)$, donde R_1 y R_2 son expresiones regulares.
6. $\mathcal{L}(R_1^*) = \bigcup_{k=0}^{\infty} \mathcal{L}(R_1)^k$, donde R_1 es una expresión regular.

Para el punto 5. y 6., definimos para dos lenguajes $L_1, L_2 \subseteq \Sigma^*$ el **producto** de L_1 y L_2:

$$\boxed{L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}}$$

Además, para un lenguaje $L \subseteq \Sigma^*$ se define la **potencia** a la $n \geq 0$:

$$\boxed{L^n = \{w_1 \cdot w_2 \ldots w_n \mid \forall i \leq n.\ w_i \in L\}}$$

La **potencia** a la 0 se define como $L^0 = \{\epsilon\}$.

Ejemplo 1.12

Se muestran a continuación lenguajes definidos por algunas ExpReg:

- $\mathcal{L}((a + b)^*) = \{a, b\}^*$
- $\mathcal{L}((a \cdot b) \cdot (b \cdot a)) = \{abba\}$
- $\mathcal{L}(a \cdot (b \cdot a) + b \cdot a + (a \cdot b) \cdot a) = \{aba, ba\}$

Definición. Decimos que R_1 es **equivalente** a R_2 si, y sólo si, $\mathcal{L}(R_1) = \mathcal{L}(R_2)$. Si R_1 es equivalente a R_2, escribiremos $R_1 \equiv R_2$.

Lema. Los operadores de unión $+$ y producto $\cdot$ son **asociativos**.

$$\boxed{\begin{array}{c}(R_1 + R_2) + R_3 \equiv R_1 + (R_2 + R_3) \\ (R_1 \cdot R_2) \cdot R_3 \equiv R_1 \cdot (R_2 \cdot R_3)\end{array}}$$

La demostración de este lema queda como ejercicio propuesto al lector.

Ejemplo 1.13

Más lenguajes definidos por algunas ExpReg:

- $\mathcal{L}(a^* \cdot b \cdot a^*) =$ todas las palabras con una sola b.
- $\mathcal{L}((a + b)^* \cdot b \cdot (a + b)^*) =$ todas las palabras con una o más $b's$.

Definición. Usamos las siguientes **abreviaciones** de expresiones regulares:

$$\boxed{\begin{array}{r}R^+ \equiv R \cdot R^* \\ R^k \equiv R \cdot \overset{k}{\cdots} \cdot R \\ R^? \equiv R + \epsilon \\ \Sigma \equiv a_1 + \ldots + a_n\end{array}}$$

para $R \in$ ExpRegs y $\Sigma = \{a_1, \ldots, a_n\}$.

Ejemplo 1.14

Más lenguajes definidos por algunas ExpReg:

- $\mathcal{L}(\Sigma^* \cdot b \cdot \Sigma^*)$ = todas las palabras con una sola b.
- $\mathcal{L}(b^* \cdot (a \cdot b)^5)$ = todas las palabras con 5 a's.
- $\mathcal{L}(a^* \cdot (b + c)^?)$ = todas las palabras de a's y terminadas en b o c.
- $\mathcal{L}((a \cdot b^+)^+)$ = todas las palabras que empiezan con a y donde cada a esta seguida de al menos una b.

1.5. Autómatas con transiciones sin lectura

Hasta ahora, hemos visto lo siguiente:

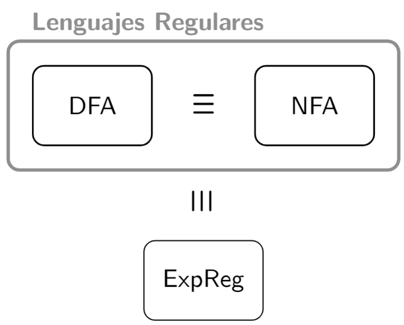


Figura 2: Mapa actual de nuestros modelos de computación

Podemos demostrar que ExpReg $\subseteq$ NFA, pero para eso necesitamos un nuevo modelo.

1.5.1. ϵ-NFA

Lo nuevo de este autómata:

1. tiene transiciones no deterministas y
2. tiene transiciones leyendo la palabra vacía ϵ:

$$p \xrightarrow{\epsilon} q$$

La importancia de un ϵ-NFA es que es un modelo **muy útil** para construir nuevos autómatas y NO agrega más poder de computación a los NFA.

Definición. Un autómata finito no-determinista con ϵ-transiciones (ϵ-NFA) es una tupla:

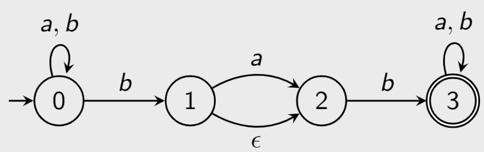
$$\boxed{\mathcal{A} = (Q, \Sigma, \Delta, I, F)}$$

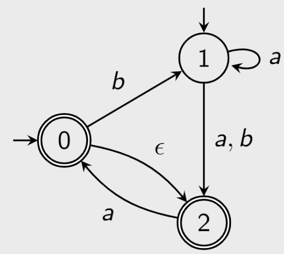
- Q es un conjunto finito de estados.
- Σ es el alfabeto del input.
- $I \subseteq Q$ es un conjunto de estados iniciales.

- $F \subseteq Q$ es el conjunto de estados finales (o aceptación).
- $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ es la relación de transición.

Ejemplo 1.15

Algunos ejemplos de ϵ-NFA's:





Para ϵ-NFA veremos una **forma alternativa** para definir las nociones de ejecución y aceptación.

Ejecución. Sea $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ un ϵ-NFA. Definimos:

- Un par $(q, w) \in Q \times \Sigma^*$ es una **configuración** de $\mathcal{A}$.
- Una configuración (q, w) es **inicial** si $q \in I$.
- Una configuración (q, w) es **final** si $q \in F$ y $w = \epsilon$.

"Intuitivamente, una configuración (q, aw) representa que $\mathcal{A}$ se encuentra en el estado q procesando la palabra aw y leyendo a".

- Se define la relación $\vdash_{\mathcal{A}} \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$ de **siguiente-paso** entre configuraciones de $\mathcal{A}$:

$$\boxed{(p, u) \vdash_{\mathcal{A}} (q, v)}$$

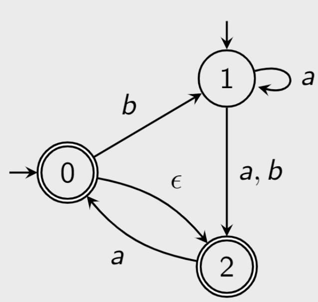
  si, y sólo si, existe $(p, c, q) \in \Delta$ con $c \in \Sigma \cup \{\epsilon\}$ tal que $u = c \cdot v$.

- Se define $\vdash^*_{\mathcal{A}}$ como la clausura **refleja** y **transitiva** de $\vdash_{\mathcal{A}}$:

$$\boxed{\begin{array}{rl} \text{para toda configuración } (q, w): & (q, w) \vdash^*_{\mathcal{A}} (q, w) \\ \text{si } (p, u) \vdash_{\mathcal{A}} (p', w) \text{ y } (p', w) \vdash^*_{\mathcal{A}} (q, v): & (p, u) \vdash^*_{\mathcal{A}} (q, v) \end{array}}$$

  Decimos que $(p, u) \vdash^*_{\mathcal{A}} (q, v)$ si uno puede ir de (p, u) a (q, v) en **0 o más pasos**.

Ejemplo 1.16



$$(1, baa) \vdash_{\mathcal{A}} (2, aa)$$

$$(2, aa) \vdash_{\mathcal{A}} (0, a) \vdash_{\mathcal{A}} (2, a) \vdash_{\mathcal{A}} (0, \epsilon)$$

$$(2, aa) \vdash^*_{\mathcal{A}} (0, \epsilon) \quad \text{y} \quad (1, baa) \vdash^*_{\mathcal{A}} (0, \epsilon)$$

Aceptación. Sea $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ un ϵ-NFA y $w \in \Sigma^*$. Decimos que $\mathcal{A}$ **acepta** w si existe una configuración **inicial** (q_0, w) y una configuración **final** (q_f, ϵ) tal que:

$$\boxed{(q_0, w) \vdash_{\mathcal{A}}^{*} (q_f, \epsilon)}$$

Además, el **lenguaje aceptado** por $\mathcal{A}$ se define como:

$$\boxed{\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ acepta } w\}}$$

Nota. Si $\mathcal{A}$ no tiene ϵ-transiciones o no-determinismo, esta es una forma **alternativa** para definir ejecución y aceptación para NFA y DFA.

1.5.2. NFA versus ϵ-NFA

Partimos enunciado el siguiente teorema:

Teorema 4

Para todo autómata finito no-determinista con ϵ-transiciones $\mathcal{A}$, existe un autómata no-determinista $\mathcal{A}'$ tal que:

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$$

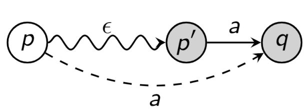
En otras palabras, NFA $\equiv$ ϵ-NFA.

Para demostrar este teorema, mostraremos como construir un autómata no-determinista a partir de un ϵ-NFA removiendo las ϵ-transiciones.

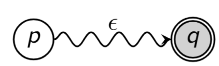
Construcción desde ϵ-NFA a NFA. Dado un ϵ-NFA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ se define el NFA:

$$\mathcal{A}^{\not\epsilon} = (Q, \Sigma, \Delta^{\not\epsilon}, I, F^{\not\epsilon})$$

- para todo $p, q \in Q$, $(p, a, q) \in \Delta^{\not\epsilon}$ si, y sólo si, existe $p' \in Q$ tal que:
  - $(p, \epsilon) \vdash_{\mathcal{A}}^{*} (p', \epsilon)$ y
  - $(p', a, q) \in \Delta$.



- $F^{\not\epsilon} = \{p \in Q \mid \exists q \in F.\ (p, \epsilon) \vdash_{\mathcal{A}}^{*} (q, \epsilon)\}$



Por definición, si $(p, a, q) \in \Delta$, entonces $(p, a, q) \in \Delta^{\not\epsilon}$ para todo $a \in \Sigma$.

Teorema 5

Dado un ϵ-NFA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ se tiene que:

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}^{\not\epsilon})$$

Demostración teorema 5. Demostrar el teorema anterior es equivalente a demostrar que para todo $p \in Q$ y $w \in \Sigma^*$:

$$\exists q \in F.\ (p, w) \vdash_{\mathcal{A}}^{*} (q, \epsilon) \quad \text{si, y sólo si,} \quad \exists q' \in F^{\not\epsilon}.\ (p, w) \vdash_{\mathcal{A}^{\not\epsilon}}^{*} (q', \epsilon)$$

De aquí, podemos **concluir** que $\mathcal{A}$ acepta w si, y sólo si, $\mathcal{A}^{\not\epsilon}$ acepta w.

Por **inducción** sobre el largo de w:

- **Caso base:** Para $w = \epsilon$:

  ($\Rightarrow$) Sea $q \in F$ tal que $(p, \epsilon) \vdash^*_{\mathcal{A}} (q, \epsilon)$. Por definición de $F^{\not\epsilon}$, se tiene que $p \in F^{\not\epsilon}$. Por lo tanto, $(p, \epsilon) \vdash^*_{\mathcal{A}^{\not\epsilon}} (p, \epsilon)$.

  ($\Leftarrow$) Sea $q' \in F^{\not\epsilon}$ tal que $(p, \epsilon) \vdash^*_{\mathcal{A}^{\not\epsilon}} (q', \epsilon)$. Como $\mathcal{A}^{\not\epsilon}$ no tiene ϵ-transiciones, entonces $p = q'$ y $p \in F^{\not\epsilon}$. Por definición de $F^{\not\epsilon}$, existe $q \in F$ tal que $(p, \epsilon) \vdash^*_{\mathcal{A}} (q, \epsilon)$.

- **Caso inductivo:** Sea $w = a \cdot u$ y $p \in Q$:

  ($\Leftarrow$) Sea $q' \in F^{\not\epsilon}$ tal que $(p, au) \vdash^*_{\mathcal{A}^{\not\epsilon}} (q', \epsilon)$. Por definición de $\vdash^*_{\mathcal{A}^{\not\epsilon}}$ existen $p' \in Q$ tal que:

  $$(p, au) \overset{(1)}{\vdash_{\mathcal{A}^{\not\epsilon}}} (p', u) \overset{(2)}{\vdash^*_{\mathcal{A}^{\not\epsilon}}} (q', \epsilon)$$

  Por (1) sabemos que $(p, au) \vdash^*_{\mathcal{A}} (p', u)$. (3)

  Como $|u| < |au|$ y por (2), por **HI** existe $q \in F$: $(p', u) \vdash^*_{\mathcal{A}} (q, \epsilon)$. (4)

  Juntando (3) y (4), tenemos que $(p, au) \vdash^*_{\mathcal{A}} (q, \epsilon)$.

  ($\Rightarrow$) Sea $q \in F$ tal que $(p, au) \vdash^*_{\mathcal{A}} (q, \epsilon)$. Por definición de $\vdash^*_{\mathcal{A}}$ existen $p', p'' \in Q$ tal que:

  $$(p, au) \overset{(1)}{\vdash^*_{\mathcal{A}}} (p', au) \overset{(2)}{\vdash_{\mathcal{A}}} (p'', u) \overset{(3)}{\vdash^*_{\mathcal{A}}} (q, \epsilon)$$

  Por (1) tenemos que $(p, \epsilon) \vdash^*_{\mathcal{A}} (p', \epsilon)$. (4)

  Por (2) tenemos que $(p', a, p'') \in \Delta$. (5)

  Por (4) y (5), sabemos que $(p, a, p'') \in \Delta^{\not\epsilon}$ y $(p, a \cdot u) \vdash_{\mathcal{A}^{\not\epsilon}} (p'', u)$. (6)

  Como $|u| < |au|$ y (3), **por HI** existe $q' \in F^{\not\epsilon}$: $(p'', u) \vdash^*_{\mathcal{A}^{\not\epsilon}} (q', \epsilon)$. (7)

  Juntando (6) y (7), tenemos que $(p, au) \vdash^*_{\mathcal{A}^{\not\epsilon}} (q', \epsilon)$. ■

Con el teorema 5 demostrado, nuestro mapa de modelos se ve así:

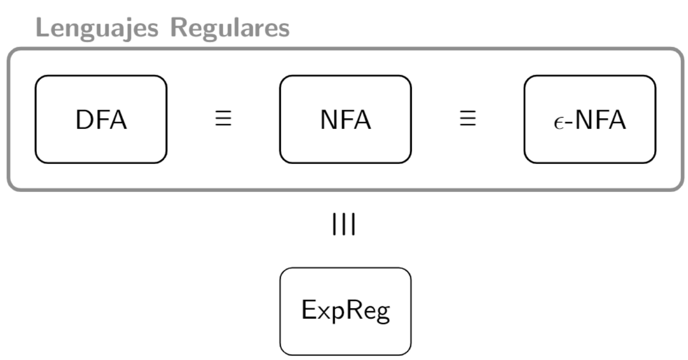


Figura 3: Mapa actual de nuestros modelos de computación

En la siguiente sección mostraremos que todos definen el mismo conjunto de lenguajes.

1.6. Teorema de Kleene

1.6.1. Desde Expresiones a Autómatas

Veremos a continuación que toda ExpReg se puede transformar en un autómata.

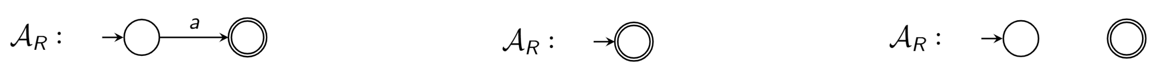
Construcción inductiva. Para cada $R \in$ ExpReg, construimos un ϵ-NFA $\mathcal{A}_R$:

$$\boxed{\mathcal{A}_R = (Q, \Sigma, \Delta, \{q_0\}, \{q_f\})}$$

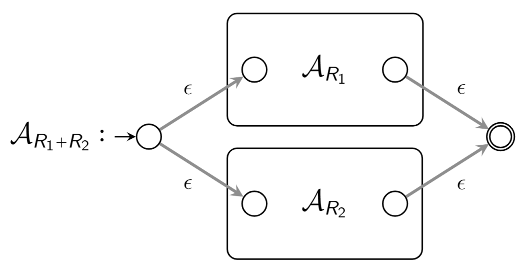
tal que $\mathcal{L}(R) = \mathcal{L}(\mathcal{A}_R)$.

Casos bases. Vemos primero los casos base de la construcción inductiva:

1. si $R = a$, para alguna letra $a \in \Sigma$:
2. si $R = \epsilon$:
3. si $R = \varnothing$:



4. si $R = (R_1 + R_2)$, donde R_1 y R_2 son expresiones regulares:



Hacemos la construcción inductiva de $R = (R_1 + R_2)$ por **inducción**. Sea $\mathcal{A}_{R_1}$ y $\mathcal{A}_{R_2}$ los ϵ-NFA para R_1 y R_2, respectivamente, tal que:

- $\mathcal{A}_{R_1} = (Q_1, \Sigma, \Delta_1, \{q_0^1\}, \{q_f^1\})$
- $\mathcal{A}_{R_2} = (Q_2, \Sigma, \Delta_2, \{q_0^2\}, \{q_f^2\})$

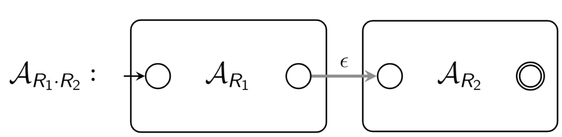
Definimos el ϵ-NFA $\mathcal{A}_{R_1+R_2} = (Q, \Sigma, \Delta, \{q_0\}, \{q_f\})$ tal que:

- $Q = Q_1 \uplus Q_2 \uplus \{q_0, q_f\}$
- $\Delta = \Delta_1 \uplus \Delta_2 \uplus \left\{ \left(q_0, \epsilon, q_0^1\right), \left(q_0, \epsilon, q_0^2\right), \left(q_f^1, \epsilon, q_f\right), \left(q_f^2, \epsilon, q_f\right) \right\}$

Proposición. Si $R = (R_1 + R_2)$, entonces $\mathcal{L}\left(R_1 + R_2\right) = \mathcal{L}\left(\mathcal{A}_{R_1+R_2}\right)$.

La demostración de esta proposición queda como ejercicio propuesto para el lector.

5. si $R = (R_1 \cdot R_2)$, donde R_1 y R_2 son expresiones regulares:



Hacemos la construcción inductiva de $R = (R_1 \cdot R_2)$ por **inducción**. Sea $\mathcal{A}_{R_1}$ y $\mathcal{A}_{R_2}$ los ϵ-NFA para R_1 y R_2, respectivamente, tal que:

- $\mathcal{A}_{R_1} = (Q_1, \Sigma, \Delta_1, \{q_0^1\}, \{q_f^1\})$
- $\mathcal{A}_{R_2} = (Q_2, \Sigma, \Delta_2, \{q_0^2\}, \{q_f^2\})$

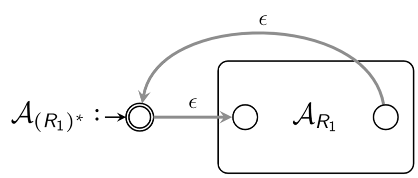
Definimos el ϵ-NFA $\mathcal{A}_{R_1 \cdot R_2} = (Q, \Sigma, \Delta, \{q_0^1\}, \{q_f^2\})$ tal que:

- $Q = Q_1 \uplus Q_2$
- $\Delta = \Delta_1 \uplus \Delta_2 \uplus \left\{ \left(q_f^1, \epsilon, q_0^2 \right) \right\}$

Proposición. Si $R = (R_1 \cdot R_2)$, entonces $\mathcal{L}\left(R_1 \cdot R_2\right) = \mathcal{L}\left(\mathcal{A}_{R_1 \cdot R_2}\right)$.

La demostración de esta proposición queda como ejercicio propuesto para el lector.

6. si $R = (R_1^*)$, donde R_1 es una expresión regular:



Hacemos la construcción inductiva de $R = (R_1^*)$ por **inducción**. Sea $\mathcal{A}_{R_1}$ el ϵ-NFA para R_1, tal que:

- $\mathcal{A}_{R_1} = (Q_1, \Sigma, \Delta_1, \{q_0^1\}, \{q_f^1\})$

Definimos el ϵ-NFA $\mathcal{A}_{(R_1^*)} = (Q, \Sigma, \Delta, \{q_0\}, \{q_0\})$ tal que:

- $Q = Q_1 \uplus \{q_0\}$
- $\Delta = \Delta_1 \biguplus \left\{ \left(q_0, \epsilon, q_0^1\right), \left(q_f^1, \epsilon, q_0 \right) \right\}$

Proposición. Si $R = (R_1^*)$, entonces $\mathcal{L}\left(R_1^*\right) = \mathcal{L}\left(\mathcal{A}_{(R_1^*)}\right)$.

La demostración de esta proposición queda como ejercicio propuesto para el lector.

Teorema 6

Para todo $R \in$ ExpReg, existe un ϵ-NFA $\mathcal{A}_R$ tal que:

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

En otras palabras, ExpReg $\subseteq$ ϵ-NFA.

1.6.2. Desde Autómatas a Expresiones

Dado un autómata finito no-determinista (que, sin pérdida de generalidad, tiene un estado inicial):

$$\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$$

Para $X \subseteq Q$ y $p, q \in Q$, considerar el conjunto:

$$\alpha_{p,q}^X \subseteq \Sigma^*$$

tal que $w = a_1 \ldots a_n \in \alpha_{p,q}^X$ si, y sólo si, existe una **ejecución**:

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} p_n$$

1. $(p_i, a_{i+1}, p_{i+1}) \in \Delta$ para todo $i \in [0, n-1]$,
2. $p_0 = p$,
3. $p_n = q$, y
4. $p_i \in X$ para todo $i \in [1, n-1]$.

"Intuitivamente, $\alpha_{p,q}^X$ es el conjunto de todas las palabras w tal que existe un ***camino*** *(i.e. ejecución) desde p a q etiquetado por w y* ***todos los estados*** *en este camino están en X, con la posible excepción de p y q".*

¿Cómo definimos $\mathcal{L}(\mathcal{A})$ en términos de $\alpha_{p,q}^X$? Establecemos el siguiente lema:

$$\mathcal{L}(\mathcal{A}) = \bigcup_{q \in F} \alpha_{q_0,p}^Q$$

Estrategia. Conocida como el algoritmo de McNaughton-Yamada:

1. Para cada $\alpha_{p,q}^X$, definir **inductivamente** una expresión regular $R_{p,q}^X$:

$$\mathcal{L}(R_{p,q}^X) = \alpha_{p,q}^X$$

2. Para $F = \{p_1, \ldots, p_k\}$ definir la **expresión regular**:

$$R_{\mathcal{A}} = R_{q_0,p_1}^Q + R_{q_0,p_2}^Q + \ldots + R_{q_0,p_k}^Q$$

3. Demostrar que:

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(R_{\mathcal{A}})$$

Definición inductiva de $R_{p,q}^X$. Tenemos que:

- **Caso base:** $X = \varnothing$

  Sea $a_1, \ldots, a_k \in \Sigma$ todas las letras tal que:

$$(p, a_i, q) \in \Delta$$

  - Si $p \neq q$, entonces:

$$R_{p,q}^{\varnothing} \overset{\text{def}}{\equiv} \begin{cases} a_1 + \cdots + a_k & \text{si } k \geq 1 \\ \varnothing & \text{si } k = 0 \end{cases}$$

  - Si $p = q$, entonces:

$$R_{p,q}^{\varnothing} \overset{\text{def}}{\equiv} \begin{cases} a_1 + \cdots + a_k + \epsilon & \text{si } k \geq 1 \\ \epsilon & \text{si } k = 0 \end{cases}$$

- **Caso general:** $X \neq \varnothing$

  Por **inducción**, suponemos que para todo $r, s \in Q$ y para todo $Y \subset X$, $R_{r,s}^Y$ es una expresión regular tal que:

$$\mathcal{L}(R_{r,s}^Y) = \alpha_{r,s}^Y$$

  Demostramos la construcción para $R_{p,q}^X$ con $p, q \in Q$. Sea $r \in X$ cualquiera:

$$R_{p,q}^X \overset{\text{def}}{\equiv} R_{p,q}^{X-\{r\}} + R_{p,r}^{X-\{r\}} \cdot \left(R_{r,r}^{X-\{r\}}\right)^* \cdot R_{r,q}^{X-\{r\}}$$

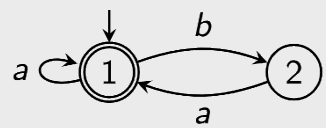
Proposición. Para todo $X \subseteq Q$ y $p, q \in Q$:

$$\mathcal{L}\left(R_{p,q}^X\right) = \alpha_{p,q}^X$$

Corolario. $\mathcal{L}(\mathcal{A}) = \mathcal{L}\left(R_{\mathcal{A}}\right)$

Ejemplo 1.17: Algoritmo MNY

Considere el autómata:



$R^{\varnothing}$	1	2
1	$a^?$	b
2	a	ϵ

$R^{\{1\}}$	1	2
1	$a^? + a^? \cdot \left(a^?\right)^* \cdot a^?$	$b + a^? \cdot \left(a^?\right)^* \cdot b$
2	$a + a \cdot \left(a^?\right)^* \cdot a^?$	$\epsilon + a \cdot \left(a^?\right)^* \cdot b$

$$\equiv$$

$R^{\{1\}}$	1	2
1	a^*	a^*b
2	a^+	$\epsilon + a^+b$

$R^{\{1,2\}}$	1	2
1	$a^* + a^*b\,(\epsilon + a^+b)^*\,a^+$	$a^*b + a^*b\,(\epsilon + a^+b)^*\,(\epsilon + a^+b)$
2	$a^+ + (\epsilon + a^+b)\,(\epsilon + a^+b)^*\,a^+$	$(\epsilon + a^+b) + (\epsilon + a^+b)\,(\epsilon + a^+b)^*\,(\epsilon + a^+b)$

$$\equiv$$

$R^{\{1,2\}}$	1	2
1	$a^*\,(ba^+)^*$	$(a^*b)\,(a^+b)^*$
2	$(a^+b)^*\,a^+$	$(a^+b)^*$

2. Propiedades de lenguajes regulares

2.1. Lema de bombeo

Supongamos que deseamos aceptar el siguiente lenguaje:

$$L = \left\{a^i b^i \mid i \geq 0\right\} = \{\epsilon, ab, aabb, aaabbb, aaaaabbbb, \ldots\}$$

con un **autómata finito determinista**. ¿Es posible? La respuesta es que no, ya que nuestros autómatas no tienen la capacidad de “contar”. Por ende, L sería un lenguaje NO regular ya que no podría ser definido por un autómata.

Enunciado. Sea $L \subseteq \Sigma^*$. Si L es **regular**, entonces:

(LB) existe un $N > 0$ tal que
para toda palabra $x \cdot y \cdot z \in L$ con $|y| \geq N$
existen palabras $u \cdot v \cdot w = y$ con $v \neq \epsilon$ tal que
para todo $i \geq 0$, $\quad x \cdot u \cdot v^i \cdot w \cdot z \in L$.

El contrapositivo del lema de bombeo nos servirá para demostrar que un lenguaje L NO es regular.

Sea $L \subseteq \Sigma^*$. Si:

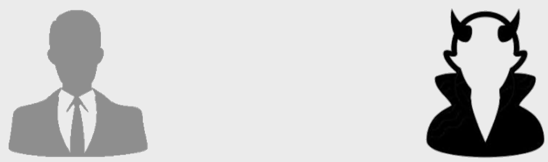
($\neg$LB) para todo $N > 0$
existe una palabra $x \cdot y \cdot z \in L$ con $|y| \geq N$ tal que
para todo $u \cdot v \cdot w = y$ con $v \neq \epsilon$
existe un $i \geq 0$, $\quad x \cdot u \cdot v^i \cdot w \cdot z \notin L$.
entonces L NO es regular.

“L **NO es** regular”	“L **es** regular”
	El escoge un $N > 0$
Uno escoge $x \cdot y \cdot z \in L$ con $\lvert y \rvert \geq N$	
	El escoge $u \cdot v \cdot w = y$ con $v \neq \epsilon$
Uno escoge $i \geq 0$	
Uno gana si $xuv^iwz \notin L$	**El gana** si $xuv^iwz \in L$

LB versión juego. *“Dado un lenguaje $L \subseteq \Sigma^*$, si **UNO** tiene una estrategia ganadora en el juego ($\neg$LB) para toda estrategia posible del demonio, entonces L **NO es** regular”.* Con **estrategia**, nos referimos a todas las movidas posibles que podría ejecutar el **demonio** (considerar todos los casos posibles de sus elecciones).

Ejemplo 2.1

Considere el lenguaje $L = \{a^i b^i \mid i \geq 0\}$:



“$a^n b^n$ **NO es** regular” — “$a^n b^n$ **es** regular”

Escojo $N > 0$

Yo escojo $\underbrace{a^N}_{x} \cdot \underbrace{b^N}_{y} \cdot \underbrace{\epsilon}_{z} \in L$

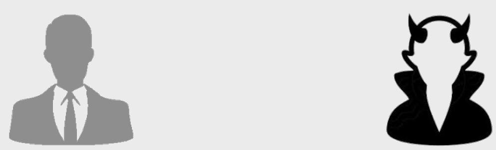
Entonces escojo $\underbrace{b^n}_{u} \cdot \underbrace{b^m}_{v} \cdot \underbrace{b^l}_{w} = \underbrace{b^N}_{y}$ con $m > 0$

Yo escojo $i = 2$

Ganamos el juego ya que con $i = 2$ estaremos bombeando más b-letras y entonces la palabra no tendrá la misma cantidad de a-letras que de b-letras ($i \neq j$), por ende, L NO es regular.

Ejemplo 2.2

Considere el lenguaje $L = \{a^n b^m \mid n \geq m\}$:



“$a^n b^m$ **NO es** regular” — “$a^n b^m$ **es** regular”

Escojo $N > 0$

Yo escojo $\underbrace{a^N}_{x} \cdot \underbrace{b^N}_{y} \cdot \underbrace{\epsilon}_{z} \in L$

Entonces escojo $\underbrace{b^j}_{u} \cdot \underbrace{b^k}_{v} \cdot \underbrace{b^l}_{w} = \underbrace{b^N}_{y}$ con $k > 0$

Yo escojo $i = 2$

Ganamos el juego ya que, nuevamente, con $i = 2$, estaremos bombeando más b-letras y entonces la palabra puede tener más b-letras que a-letras ($n < m$), y así L NO es regular.

Ejemplo 2.3

Considere el lenguaje $L = \{w \cdot w \mid w \in \{a,b\}^*\}$


"L **NO es** regular" — "L **es** regular"

Escojo $N > 0$

Yo escojo $\underbrace{a^N b}_{x} \cdot \underbrace{a^N}_{y} \cdot \underbrace{b}_{z} \in L$

Entonces escojo $\underbrace{a^j}_{u} \cdot \underbrace{a^k}_{v} \cdot \underbrace{a^l}_{w} = \underbrace{a^N}_{y}$ con $k > 0$

Yo escojo $i = 0$

Ganamos el juego ya que con la elección de $i = 0$ estamos haciendo que una de las mitades de la palabra sea distinta a su otra mitad, por ende, L NO es regular.

Ejemplo 2.4

Considere el lenguaje $L = \{a^{2^n} \mid n > 0\}$


"a^{2^n} **NO es** regular" — "a^{2^n} **es** regular"

Escojo $N > 0$

Yo escojo $\underbrace{a^{2^n - N}}_{x} \cdot \underbrace{a^N}_{y} \cdot \underbrace{\epsilon}_{z} \in L$ con $N < 2^n$

Entonces escojo $\underbrace{a^j}_{u} \cdot \underbrace{a^k}_{v} \cdot \underbrace{a^l}_{w} = \underbrace{a^N}_{y}$ con $k > 0$

Yo escojo $i = 2$

Ganamos el juego ya que con la elección de $i = 2$, tenemos que en la elección de y tendremos una mayor cantidad de a-letras bombeadas y se romperá el equilibrio $2^N - N + N$, por ende, L NO es regular.

2.2. Minimización de autómatas

¿Cómo minimizamos un autómata finito?

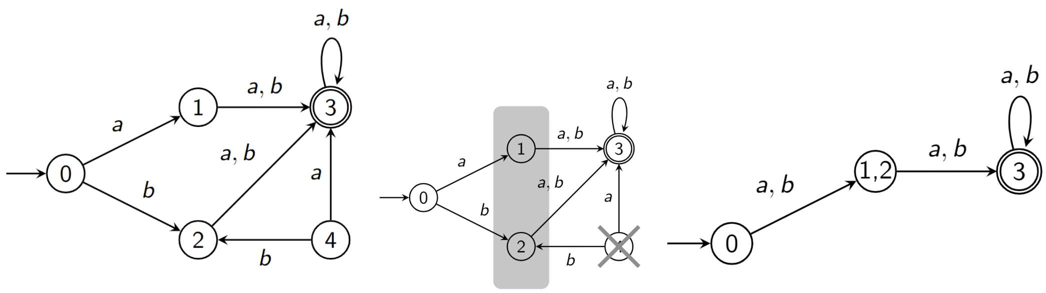
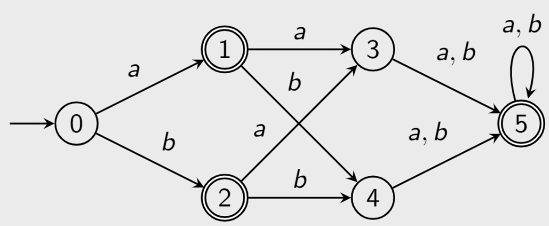


Figura 4: Idea de minimización

1. Eliminar estados inaccesibles.
   - Fácil de realizar y no cambia el lenguaje del autómata finito.
2. Colapsar estados "equivalentes".
   - ¿Cómo sabemos cuáles estados colapsar y cúales no?

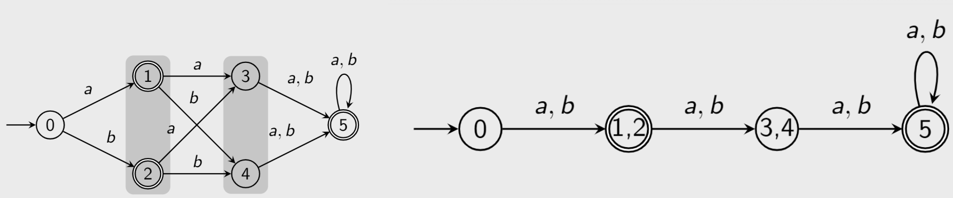
Ejemplo 2.5

Considere el siguiente autómata:



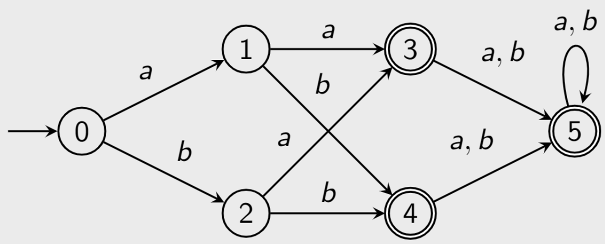
Podemos:

- Colapsar estados 1 y 2.
- Colapsar estados 3 y 4.



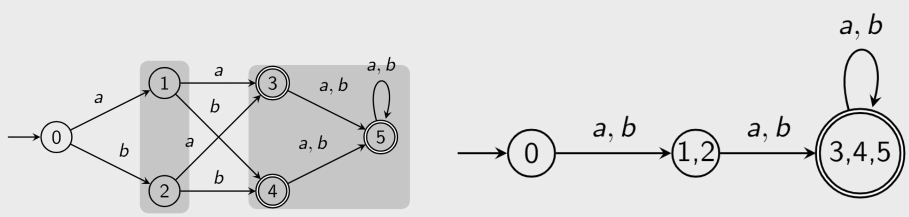
Ejemplo 2.6

Considere el siguiente autómata:



Podemos:

- Colapsar estados 1 y 2.
- Colapsar estados 3, 4 y 5.



2.2.1. Colapsar estados

Definición. Sea $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un DFA.
Se define la **función de transición extendida** $\hat{\delta} : Q \times \Sigma^* \to Q$ inductivamente como:

$$\boxed{\begin{array}{rcl} \hat{\delta}(q, \epsilon) & \overset{\text{def}}{\equiv} & q \\ \hat{\delta}(q, w \cdot a) & \overset{\text{def}}{\equiv} & \delta(\hat{\delta}(q, w), a) \end{array}}$$

Definición. Decimos que p y q son **indistingibles** ($p \approx_{\mathcal{A}} q$) si:

$$\boxed{p \approx_{\mathcal{A}} q \quad \text{ssi} \quad (\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F), \text{ para todo } w \in \Sigma^*.}$$

Decimos que p y q son **distingibles** si NO son indistingibles ($p \not\approx_{\mathcal{A}} q$).

Recordatorio relaciones de equivalencia. Una relación $\approx_R$ sobre un conjunto X se dice de **equivalencia** si es:

- **Refleja:** $\forall p \in X.\ p \approx_R p$
- **Simétrica:** $\forall p, q \in X.$ si $p \approx_R q$ entonces $q \approx_R p$.
- **Transitiva:** $\forall p, q, r \in X.$ si $p \approx_R q$ y $q \approx_R r$, entonces $p \approx_R r$.

Para un elemento $p \in X$ se define su **clase de equivalencia** según $\approx_R$ como:

$$[p]_{\approx_R} = \{q \mid q \approx_R p\}$$

Una función $f : X \to X$ se dice **bien definida** sobre $\approx_R$ si:

$$p \approx_R q \quad \text{entonces} \quad f(p) \approx_R f(q)$$

Propiedades de $\approx_{\mathcal{A}}$. Tenemos que:

- $\approx_{\mathcal{A}}$ es una **relación de equivalencia**, es decir, es refleja, simétrica y transitiva.
- Cada estado $p \in Q$ esta en exactamente una clase de equivalencia:

$$[p]_{\approx_{\mathcal{A}}} = \{q \mid q \approx_{\mathcal{A}} p\}$$

- Para todo $a \in \Sigma$ la función $\delta(\cdot, a) : Q \to Q$ esta **bien definida** sobre $\approx_{\mathcal{A}}$:

$$p \approx_{\mathcal{A}} q \quad \text{entonces} \quad \delta(p, a) \approx_{\mathcal{A}} \delta(q, a)$$

El autómata cuociente. Para un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ se define el DFA:

$$\boxed{\mathcal{A}/\approx = (Q_{\approx}, \Sigma, \delta_{\approx}, q_{\approx}, F_{\approx})}$$

- $Q_{\approx} = \{[p]_{\approx_{\mathcal{A}}} \mid p \in Q\}$
- $\delta_{\approx}([p]_{\approx_{\mathcal{A}}}, a) = [\delta(p, a)]_{\approx_{\mathcal{A}}}$
- $q_{\approx} = [q_0]_{\approx_{\mathcal{A}}}$
- $F_{\approx} = \{[p]_{\approx_{\mathcal{A}}} \mid p \in F\}$

Teorema 7

Para todo autómata finito determinista $\mathcal{A}$ se cumple que:

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}/\approx)$$

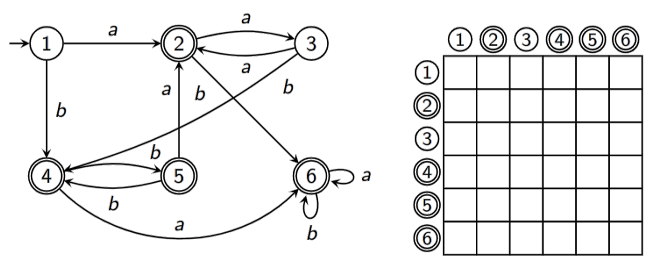
Demostración. PENDIENTE. Pero se invita al lector a hacerla <3.

2.2.2. Algoritmo de minimización

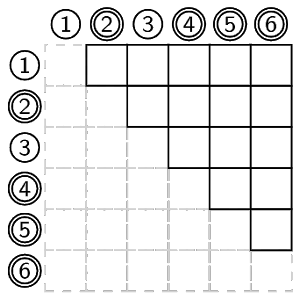
El objetivo es buscar los pares de estados que son **distingibles**:

1. Construya una tabla con los pares $\{p, q\}$ inicialmente sin marcar.
2. Marque $\{p, q\}$ si $p \in F$ y $q \notin F$ o viceversa.
3. Repita este paso hasta que no hayan más cambios:
   - Si $\{p, q\}$ no están marcados y $\{\delta(p, a), \delta(q, a)\}$ estan marcados para algún $a \in \Sigma$, entonces marque $\{p, q\}$.
4. Al terminar, $p \not\approx_{\mathcal{A}}$ si, y sólo si, la entrada $\{p, q\}$ está marcada.

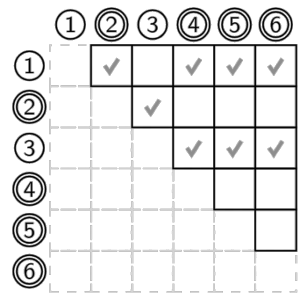
Veamos como funciona el algoritmo. Considere el siguiente autómata y una tabla que relacione todos los pares de estados:



1. Construya una tabla con los pares $\{p, q\}$ inicialmente sin marcar.

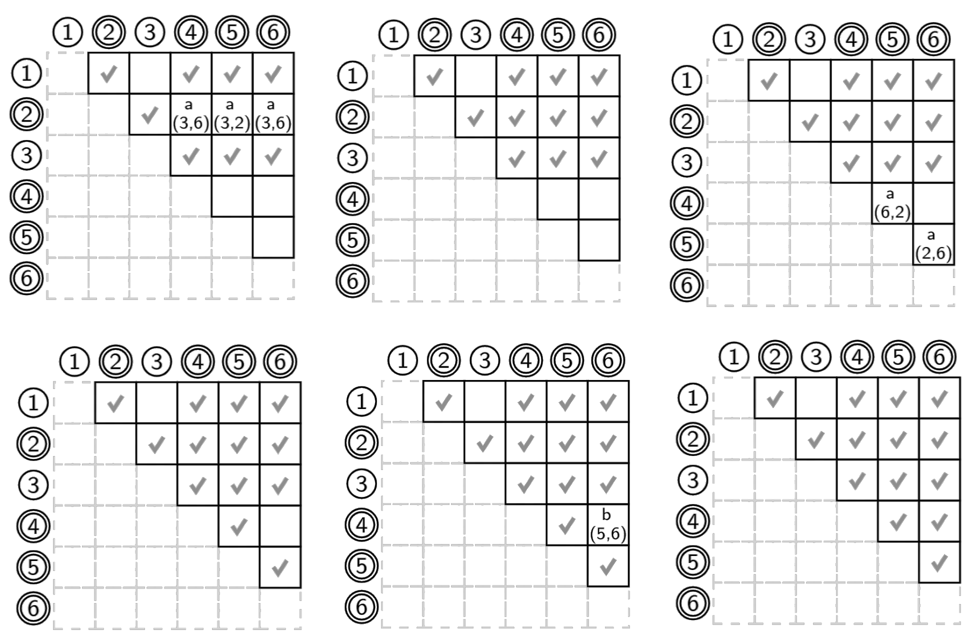


2. Marque $\{p, q\}$ si $p \in F$ y $q \notin F$ o viceversa.



3. Repita este paso hasta que no hayan más cambios:

   - Si $\{p, q\}$ no están marcados y $\{\delta(p, a), \delta(q, a)\}$ estan marcados para algún $a \in \Sigma$, entonces marque $\{p, q\}$.



4. Al terminar, $p \not\approx_{\mathcal{A}}$ si, y sólo si, la entrada $\{p, q\}$ está marcada.

Así, vemos que los pares indistingibles son todas las entradas **NO** marcadas.

2.3. Teorema de Myhill-Nerode

La sección anterior deja muchas incógnitas:

1. ¿Cómo sabemos si el autómata del algoritmo es un **mínimo**?
2. Dado L, ¿existe un **único** autómata mínimo?
3. Dado un $\mathcal{A}$, ¿es posible **construir** un autómata mínimo equivalente?

En esta sección, demostraremos que:

- El autómata con el mínimo de estados es **único**.
- El algoritmo de minimización **siempre** construye el autómata mínimo.

Estrategia. Para demostrar lo dicho anteriormente, seguimos los siguientes pasos:

1. Desde un DFA $\mathcal{A}$, definiremos una relación de equivalencia (RE) $\equiv_{\mathcal{A}}$ entre palabras en Σ^*.
2. Desde una RE $\equiv$ entre palabras, construiremos un DFA $\mathcal{A}_{\equiv}$.
3. A partir de un lenguaje L, definiremos una RE $\equiv_L$.
4. $\mathcal{A}_{\equiv_L}$ define el autómata con la **menor cantidad de estados**.
5. $\mathcal{A}_{\equiv_L}$ es equivalente al resultado de nuestro **algoritmo de minimización**.

2.3.1. Relaciones de Myhill-Nerode

Sea $L \subseteq \Sigma^*$ cualquier lenguaje.

Definición. Una relación de equivalencia $\equiv$ en Σ^* es de **Myhill-Nerode** para L si:

1. $\equiv$ es una **congruencia por la derecha**.
2. $\equiv$ **refina** L.
3. El número de clases de equivalencia de $\equiv$ es **finita**.

A partir de una relación $\equiv$ de Myhill-Nerode podemos construir un DFA $\mathcal{A}_{\equiv}$.

$$\boxed{\begin{array}{ccc} \mathcal{A} & \Rightarrow & \equiv_{\mathcal{A}} \\ \equiv & \Rightarrow & \mathcal{A}_{\equiv} \end{array}}$$

Construcción del DFA $\mathcal{A}_{\equiv}$. Dada una relación de Myhill-Nerode $\equiv$ para $L \subseteq \Sigma^*$, definimos el autómata:

$$\boxed{\mathcal{A}_{\equiv} = (Q_{\equiv}, \Sigma, \delta_{\equiv}, q_{\equiv}, F_{\equiv})}$$

- $Q_{\equiv} = \{[w]_{\equiv} \mid w \in \Sigma^*\}$
- $q_{\equiv} = [\epsilon]_{\equiv}$
- $F_{\equiv} = \{[w]_{\equiv} \mid w \in L\}$
- $\delta_{\equiv}([w]_{\equiv}, a) = [wa]_{\equiv}$

Teorema 8

Cada cualquier $L \subseteq \Sigma^*$, tenemos que

$$\mathcal{L}(\mathcal{A}_\equiv) = L$$

Podemos establecer que $\mathcal{A} \Rightarrow \equiv_\mathcal{A}$ y $\equiv \Rightarrow \mathcal{A}_\equiv$ son procesos inversos, conclusión que se ilustra en el siguiente teorema.

Teorema 9

1. *Si $\mathcal{A}$ es un DFA que acepta L y si construimos:*

$$\mathcal{A} \quad \Rightarrow \quad \equiv_\mathcal{A} \quad \Rightarrow \quad \mathcal{A}_{\equiv_\mathcal{A}}$$

*entonces $\mathcal{A}$ es **isomorfo** (“equivalente”) a $\mathcal{A}_{\equiv_\mathcal{A}}$.*

2. *Si $\equiv$ es una relación de Myhill-Nerode para L y si construimos:*

$$\equiv \quad \Rightarrow \quad \mathcal{A}_\equiv \quad \Rightarrow \quad \equiv_{\mathcal{A}_\equiv}$$

*entonces la relación $\equiv$ es **equivalente** a $\equiv_{\mathcal{A}_\equiv}$.*

La demostración de ambos teoremas queda propuesto como ejercicio al lector.

2.3.2. Camino al teorema

Antes de enunciar el Teorema de Myhill-Nerode, debemos aún mencionar algunas definiciones.

Definición. Dado un lenguaje $L \subseteq \Sigma^*$, se define la relación de equivalencia $\equiv_L$ como:

$$\boxed{u \equiv_L v \quad \text{ssi} \quad (u \cdot w \in L \Leftrightarrow v \cdot w \in L) \quad \forall w \in \Sigma^*}$$

Ejemplo 2.7

Sea $L = (ab)^*$. Algunas clases de equivalencia para L son:

- $[\epsilon]_{\equiv_L} = \{\epsilon, ab, abab, ababab, \ldots\}$.
- $[a]_{\equiv_L} = \{a, aba, ababa, abababa, \ldots\}$.
- $[b]_{\equiv_L} = \{b, bb, ba, abb, \ldots\}$

Propiedades. $\equiv_L$ se caracteriza por:

1. Ser una **congruencia por la derecha**:

$$u \equiv_L v \text{ entonces } u \cdot w \equiv_L v \cdot w \quad \forall w \in \Sigma^*$$

2. Refinar a L:

$$u \equiv_L v \text{ entonces } (u \in L \Leftrightarrow v \in L)$$

3. Si $\equiv$ es una congruencia por la derecha y refina L, entonces $\equiv$ **refina** a $\equiv_L$:

$$u \equiv v \text{ entonces } u \equiv_L v$$

Con todo lo anterior, estamos en condiciones de enunciar el teorema.

Teorema 10

Sea $L \subseteq \Sigma^*$. Las siguientes propiedades son equivalentes:

1. *L es **regular**.*
2. *Existe una **relación de Myhill-Nerode** para L.*
3. *La relación $\equiv_L$ tiene una cantidad **finita** de clases de equivalencia.*

Demostración teorema 10. Del punto 1 al 2, tenemos que si L es regular, entonces:

- existe un autómata finito $\mathcal{A}$ tal que $L = \mathcal{L}(\mathcal{A})$.
- $\equiv_{\mathcal{A}}$ es una relación de Myhill-Nerode para L.

Del punto 2 al 3, sea $\equiv$ una relación de Myhill-Nerode para L, entonces:

- $\equiv$ tiene una cantidad finita de clases de equivalencia.
- $\equiv_L$ tiene una cantidad finita de clases de equivalencia.

Del punto 3 al 1, si $\equiv_L$ tiene una cantidad **finita** de clases de equivalencia, entonces:

- $\equiv_L$ es una relación de Myhill-Nerode para L.
- $\mathcal{A}_{\equiv_L}$ es un autómata finito para L. ■

Conclusiones del teorema. Tenemos que:

1. $\equiv_L \Rightarrow \mathcal{A}_{\equiv_L}$ produce el autómata con la menor cantidad de estados.
2. Todo autómata $\mathcal{A}$ tal que $\equiv_{\mathcal{A}} = \equiv_L$ son **isomorfos** ("equivalentes").
3. El **algoritmo de minimización** produce un autómata isomorfo $\mathcal{A}_{\equiv_L}$.

Demostración punto 3. Sea $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un autómata que acepta L ya **minimizado**:

$$\begin{aligned}
u \equiv_L v \quad &\Leftrightarrow \quad (u \cdot w \in L \Leftrightarrow v \cdot w \in L) \quad \forall w \in \Sigma^* \\
&\Leftrightarrow \quad \left(\hat{\delta}(q_0, u \cdot w) \in F \Leftrightarrow \hat{\delta}(q_0, v \cdot w) \in F\right) \quad \forall w \in \Sigma^* \\
&\Leftrightarrow \quad \left(\hat{\delta}\left(\hat{\delta}(q_0, u), w\right) \in F \Leftrightarrow \hat{\delta}\left(\hat{\delta}(q_0, v), w\right) \in F\right) \quad \forall w \in \Sigma^* \\
&\Leftrightarrow \quad \hat{\delta}(q_0, u) \approx_{\mathcal{A}} \hat{\delta}(q_0, v) \\
&\Leftrightarrow \quad \hat{\delta}(q_0, u) = \hat{\delta}(q_0, v) \\
&\Leftrightarrow \quad u \equiv_{\mathcal{A}} v
\end{aligned}$$

■

2.4. Autómatas en dos direcciones

3. Algoritmos para lenguajes regulares

3.1. Evaluación de expresiones regulares

3.2. Transductores

3.3. Análisis léxico

3.4. Algoritmo de Knuth-Morris-Prat

4. Lenguajes libres de contexto

4.1. Gramáticas libres de contexto

4.2. Simplificación de gramáticas

4.3. Forma normal de Chomsky

4.4. Lema de bombeo para lenguajes libres de contexto

4.5. Algoritmo CKY

5. Algoritmos para lenguajes libres de contexto

5.1. Autómatas apiladores

5.1.1. Versión normal

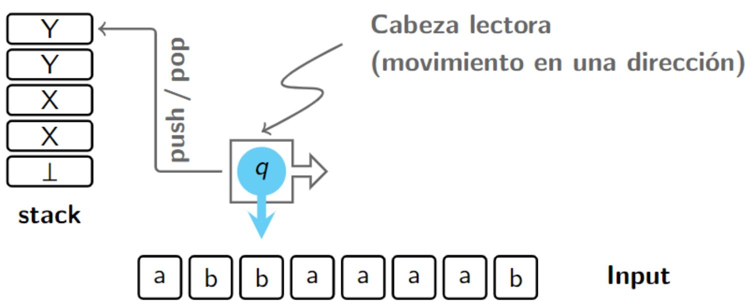


Figura 5: Idea de un autómata apilador

Definición. Un autómata apilador (*PushDown Automata*, PDA) es una estructura:

$$\boxed{\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)}$$

- Q es un conjunto finito de **estados**.
- Σ es el alfabeto del **input**.
- $q_0 \in Q$ es el estado **inicial**.
- F es el conjunto de estados **finales**.
- Γ es el alfabeto de **stack**.
- $\perp \in \Gamma$ es el símbolo **inicial del stack** (fondo).
- $\Delta \subseteq (Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma) \times (Q \times \Gamma^*)$ es una relación finita de transición.

Intuitivamente, la transición:

$$\boxed{\Big((p, a, A), (q, B_1 B_2 \cdots B_k)\Big) \in \Delta}$$

si el autómata apilador está:

- en el estado p, leyendo a, y en el tope del stack hay una A,

entonces:

- cambia al estado q, y modifico el tope A por $B_1 B_2 \cdots B_k$.

Intuitivamente, la transición **en vacío**:

$$\boxed{\Big((p, \epsilon, A), (q, B_1 B_2 \cdots B_k)\Big) \in \Delta}$$

si el autómata apilador está:

- en el estado p, *sin lectura de una letra*, y en el tope del stack hay una A,

entonces:

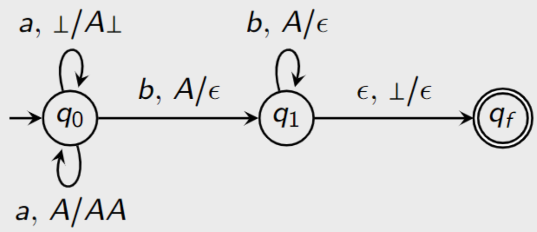
- cambia al estado q, y modifico el tope A por $B_1 B_2 \cdots B_k$.

Ejemplo 5.1

$$\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, \{q_f\})$$

- $Q = \{q_0, q_1, q_f\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, \perp\}$ y Δ:

$$\begin{array}{ll}
(q_0, a, \perp, q_0, A\perp) & q_0 \perp \xrightarrow{a} q_0 A \perp \\
(q_0, a, A, q_0, AA) & q_0 A \xrightarrow{a} q_0 AA \\
(q_0, b, A, q_1, \epsilon) & q_0 A \xrightarrow{b} q_1 \\
(q_1, b, A, q_1, \epsilon) & q_1 A \xrightarrow{b} q_1 \\
(q_1, \epsilon, \perp, q_f, \epsilon) & q_1 \perp \xrightarrow{\epsilon} q_f
\end{array}$$



Notación. Dada una palabra $A_1 A_2 \ldots A_k \in \Gamma^+$ decimos que:

- $A_1 A_2 \ldots A_k$ es un stack (contenido),
- A_1 es el **tope** del stack y
- $A_2 \ldots A_k$ es la **cola** del stack.

Definición. Una **configuración** de $\mathcal{P}$ es una tupla $(q \cdot \gamma, w) \in (Q \cdot \Gamma^*, \Sigma^*)$ tal que:

- q es el estado actual.
- γ es el contenido del stack.
- w es el contenido del input.

Decimos que una configuración:

$$\boxed{(q \cdot \gamma, w) \in (Q \cdot \Gamma^*, \Sigma^*)}$$

- es **inicial** si $q \cdot \gamma = q_0 \cdot \perp$.
- es **final** si $q \cdot \gamma = q_f \cdot \epsilon$ con $q_f \in F$ y $w = \epsilon$.

Definición. Se define la relación $\vdash_{\mathcal{P}}$ de **siguiente-paso** entre configuraciones de $\mathcal{P}$:

$$\boxed{(q_1 \cdot \gamma_1, w_1) \quad \vdash_{\mathcal{P}} \quad (q_2 \cdot \gamma_2, w_2)}$$

si, y sólo si, existe una transición $(q_1, a, A, q_2, \alpha) \in \Delta$ y $\gamma \in \Gamma^*$ tal que:

- $w_1 = a \cdot w_2$

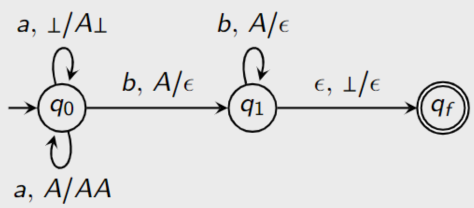
- $\gamma_1 = A \cdot \gamma$
- $\gamma_2 = \alpha \cdot \gamma$

Se define $\vdash^*_{\mathcal{P}}$ como la clausura **refleja** y **transitiva** de $\vdash_{\mathcal{P}}$. En otras palabras:

$(q_1\gamma_1, w_1) \vdash^*_{\mathcal{P}} (q_2\gamma_2, w_2)$ si uno puede ir de $(q_1\gamma_1, w_1)$ a $(q_2\gamma_2, w_2)$ en 0 o más pasos.

Ejemplo 5.2

Para la palabra $w = aaabbb$, tenemos la ejecución:



$q_0 \bot$	
$q_0\ A \bot$	a
$q_0\ A\ A \bot$	a
$q_0\ A\ A\ A \bot$	a
$q_1\ A\ A \bot$	b
$q_1\ A \bot$	b
$q_1 \bot$	b
q_f	ϵ

Definiciones. $\mathcal{P}$ **acepta** w si, y sólo si, $(q_0 \bot, w) \vdash^*_{\mathcal{P}} (q_f, \epsilon)$ para algún $q_f \in F$.
El **lenguaje aceptado** por $\mathcal{P}$ se define como:

$$\mathcal{L}(\mathcal{P}) = \{w \in \Sigma^* \| \ \mathcal{P} \text{ acepta } w\}$$

Ejemplo 5.3

El lenguaje aceptado por el PDA utilizado en los ejemplos anteriores es $\mathcal{L}(\mathcal{P}) = \{a^n b^n \mid n \geq 0\}$.

5.1.2. Versión alternativa

Esta definición de autómata apilador es poco común pero trae algunas ventajas:

- Es un modelo que ayuda a entender mejor los algoritmos de evaluación para gramáticas.
- Es un modelo menos estándar pero mucho más sencillo.
- Al profe Cristian le gustó y lo encontró interesante.

Definición. Un **PDA alternativo** es una estructura:

$$\mathcal{D} = (Q, \Sigma, \Delta, q_0, F)$$

- Q es un conjunto finito de **estados**.

- Σ es el alfabeto del **input**.
- $q_0 \in Q$ es el estado **inicial**.
- F es el conjunto de estados **finales**.
- $\Delta \subseteq Q^+ \times (\Sigma \cup \{\epsilon\}) \times Q^*$ es una **relación finita de transición**.

Intuitivamente, la transición:

$$\boxed{\Big(A_1 \dots A_i, a, B_1 \dots B_j\Big) \in \Delta}$$

si el autómata apilador tiene:

- $A_1 \dots A_i$ en el tope del stack y leyendo a,

entonces:

- cambia el tope $A_1 \dots A_i$ por $B_1 \dots B_j$.

En este tipo de autómata apilador, **no hay diferencia** entre estados y alfabeto del stack.

Definición. Una **configuración** de $\mathcal{D}$ es una tupla

$$\boxed{(q_1 \dots q_k, w) \in (Q^+, \Sigma^*)}$$

tal que:

- $q_1 \dots q_k$ es el contenido del stack con q_1 el tope del stack.
- w es el contenido del input.

Decimos que una configuración:

- (q_0, w) es **inicial**.
- (Q_f, ϵ) es **final** si $q_f \in F$.

Definición. Se define la relación $\vdash_{\mathcal{D}}$ de **siguiente-paso** entre configuraciones de $\mathcal{D}$:

$$\boxed{(\gamma_1, w_1) \quad \vdash_{\mathcal{D}} \quad (\gamma_2, w_2)}$$

si, y sólo si, existe una transición $(\alpha, a, \beta) \in \Delta$ y $\gamma \in \Gamma^*$ tal que:

- $w_1 = a \cdot w_2$
- $\gamma_1 = \alpha \cdot \gamma$
- $\gamma_2 = \beta \cdot \gamma$

Se define $\vdash^*_{\mathcal{D}}$ como la clausura **refleja** y **transitiva** de $\vdash_{\mathcal{D}}$.

Definiciones. $\mathcal{D}$ **acepta** w si, y sólo si, $(q_0, w) \vdash^*_{\mathcal{D}} (q_f, \epsilon)$ para algún $q_f \in F$. Además, el **lenguaje aceptado** por $\mathcal{D}$ se define como:

$$\boxed{\mathcal{L}(\mathcal{D}) = \{w \in \Sigma^* \| \ \mathcal{D} \text{ acepta } w\}}$$

Ejemplo 5.4

$$\mathcal{D} = (Q, \{a, b\}, \Delta, q_0, F)$$

- $Q = \{\perp, q_0, q_1, q_f\}$ y Δ:

$(\perp, a, q_0\perp)$	$\perp \xrightarrow{a} q_0\perp$
(q_0, a, q_0q_0)	$q_0 \xrightarrow{a} q_0q_0$
(q_0, b, q_1)	$q_0 \xrightarrow{a} q_1$
(q_1q_0, b, q_1)	$q_1q_0 \xrightarrow{b} q_1$
$(q_1\perp, \epsilon, q_f)$	$q_1\perp \xrightarrow{b} q_f$

$\perp$	
$q_0 \perp$	a
$q_0\ q_0 \perp$	a
$q_0\ q_0\ q_0 \perp$	a
$q_1\ q_0\ q_0 \perp$	b
$q_1\ q_0 \perp$	b
$q_1 \perp$	b
q_f	ϵ

$$\mathcal{L}(\mathcal{D}) = \{a^n b^n \mid n \geq 1\}$$

Teorema 11

Para todo autómata apilador $\mathcal{P}$ existe un autómata apilador alternativo $\mathcal{D}$, y viceversa, tal que:

$$\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{D})$$

El teorema anterior nos dice que podemos usar ambos modelos de manera **equivalente**.

5.2. Autómatas apiladores vs gramáticas libres de contexto

¿En qué se parecen CFG a PDA?

Gramáticas libre de contexto

- Terminales (Σ)
- Variables (V)
- Producciones
- $A \to \gamma$
- $A \to a$
- Derivaciones
- Árbol de derivación

Autómatas apiladores

- Alfabeto input (Σ)
- Alfabeto stack (Γ)
- Transiciones
- $pA \xrightarrow{c} q\gamma$
- $pA \xrightarrow{c} q$
- Ejecuciones
- Stack

Figura 6: Gramáticas vs Autómatas apiladores

Teorema 12

*Todo **lenguaje libre de contexto** puede ser descrito equivalentemente por:*

- *Una gramática libre de contexto (**CFG**).*
- *Un autómata apilador (**PDA**).*

5.2.1. Desde CFG a PDA

Partimos enunciado un teorema:

Teorema 13

Para toda gramática libre de contexto $\mathcal{G}$, existe un ***autómata apilador alternativo*** *$\mathcal{D}$, tal que:*

$$\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{D})$$

Construcción $\mathcal{D}$ desde $\mathcal{G}$. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG. Construimos un PDA alternativo $\mathcal{D}$ que acepta $\mathcal{L}(\mathcal{G})$:

$$\boxed{\mathcal{D} = \Big(V \cup \Sigma \cup \{q_0, q_f\}, \Sigma, \Delta, q_0, \{q_f\}\Big)}$$

La relación de transición Δ se define como:

$$\begin{array}{rclcl} \Delta & = & \{(q_0, \epsilon, S \cdot q_f)\} & \cup & \\ & & \{(X, \epsilon, \gamma) \mid X \to \gamma \in P\} & \cup & \textbf{(Expandir)} \\ & & \{(a, a, \epsilon) \mid a \in \Sigma\} & & \textbf{(Reducir)} \end{array}$$

Demostración $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{D})$. Debemos demostrar dos direcciones: $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{D})$ y $\mathcal{L}(\mathcal{D}) \subseteq \mathcal{L}(\mathcal{G})$.

Demostración $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{D})$. Para cada $w \in \mathcal{L}(\mathcal{G})$ debemos encontrar una ejecución de aceptación de $\mathcal{D}$ sobre w. ¿Cómo encontramos esta ejecución? La idea es que para cada árbol de derivación $\mathcal{T}$ de $\mathcal{G}$ sobre w, construimos una ejecución de $\mathcal{D}$ sobre w que recorre el árbol $\mathcal{T}$ **en profundidad** (DFS). Por tanto, debemos usar **inducción** sobre la altura del árbol $\mathcal{T}$.

Hipótesis de inducción. Para todo árbol de derivación $\mathcal{T}$ de $\mathcal{G}$ con **altura** h tal que:

- la raíz de $\mathcal{T}$ es X, y
- $\mathcal{T}$ produce la palabra w

entonces $(X \cdot \gamma, w) \vdash^*_{\mathcal{D}} (\gamma, \epsilon)$ para todo $\gamma \in Q^+$.

Caso base: $h = 1$. Si $\mathcal{T}$ tiene altura 1, entonces:

- $\mathcal{T}$ produce la palabra $w = a$ para algún $a \in \Sigma$ y
- $\mathcal{T}$ consiste de un nodo X y un hijo a con $X \to a$.

Entonces para todo $\gamma \in Q^+$:

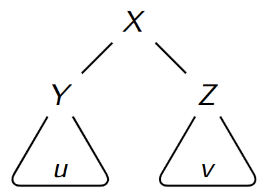
$$(X \cdot \gamma, a) \vdash_{\mathcal{D}} (a \cdot \gamma, a) \vdash_{\mathcal{D}} (\gamma, \epsilon)$$

es una ejecución de $\mathcal{D}$ sobre a.

Caso inductivo: $h = n$. Suponemos que el árbol de derivación $\mathcal{T}$ de $\mathcal{G}$ tiene **altura** n tal que:

- la raíz de $\mathcal{T}$ es X, y
- $\mathcal{T}$ produce la palabra w.

Sin pérdida de generalidad, suponga que $\mathcal{T}$ es de la forma:



donde $w = u \cdot v$ y $X \to YZ$. Por HI, se tiene que para todo $\gamma_1, \gamma_2 \in Q^+$:

$$(Y \cdot \gamma_1, u) \vdash_{\mathcal{D}}^* (\gamma_1, \epsilon)$$
$$(Z \cdot \gamma_2, v) \vdash_{\mathcal{D}}^* (\gamma_2, \epsilon)$$

Para $\gamma \in Q^+$ **construimos** la siguiente ejecución de $\mathcal{D}$ sobre $w = uv$:

$$(X \cdot \gamma, uv) \vdash_{\mathcal{D}} (YZ \cdot \gamma, uv) \vdash_{\mathcal{D}}^* (Z \cdot \gamma, v) \vdash_{\mathcal{D}}^* (\gamma, \epsilon)$$

■

La demostración de $\mathcal{L}(\mathcal{D}) \subseteq \mathcal{L}(\mathcal{G})$ se deja como ejercicio propuesto al lector.

5.2.2. Desde PDA a CFG

Partimos enunciando el siguiente teorema:

Teorema 14

Para todo autómata apilador $\mathcal{P}$, existe una gramática libre de contexto $\mathcal{G}$ tal que:

$$\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{G})$$

Demostración $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{G})$. Sea $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$ un PDA (normal). Los pasos a seguir son:

1. Convertir $\mathcal{P}$ a un PDA $\mathcal{P}'$ con **UN solo estado**.
2. Convertir $\mathcal{P}'$ a una gramática libre de contexto $\mathcal{G}$.

Paso 1. Sea $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$ un PDA. Podemos analizar:

- ¿Por qué NO necesitamos la información de los estados?
- ¿Cómo guardamos la información de los estados en el stack?

Esto conlleva a la siguiente pregunta: *Si el PDA está en el estado p y en el tope del stack hay una A, ¿a cuál estado llegaré al remover A del stack?*

La solución a esta pregunta es que podemos **adivinar** (no-determinismo) el estado que vamos a llegar cuando removamos A del stack.

Sin pérdida de generalidad, podemos asumir que

1. Todas las transiciones son de la forma:

$$qA \xrightarrow{c} pB_1B_2 \quad \text{o} \quad qA \xrightarrow{c} p\epsilon$$

   con $c \in (\Sigma \cup \{\epsilon\})$.

2. Existe $q_f \in Q$ tal que si $w \in \mathcal{L}(\mathcal{P})$ entonces:

$$(q_0\bot, w) \vdash_{\mathcal{D}}^{*} (q_f, \epsilon)$$

Estos dos puntos nos aseguran que siempre llegamos al **mismo estado** q_f. Luego, construimos el autómata apilador $\mathcal{P}'$ con **un solo estado**:

$$\mathcal{P}' = (\{q\}, \Sigma, \Gamma', \Delta', \{q\}, \bot', \{q\})$$

- $\Gamma' = Q \times \Gamma \times Q$.
  "$(p, A, q) \in \Gamma'$ si desde p leyendo A en el tope del stack llegamos a q al hacer pop de A".
- $\bot' = (q_0, \bot, q_f)$.
  "El autómata parte en q_0 y al hacer pop de $\bot$ llegará a q_f".
- Si $pA \xrightarrow{c} p'B_1B_2 \in \Delta$ con $c \in (\Sigma \cup \{\epsilon\})$, entonces **para todo** $p_1, p_2 \in Q$:

$$q\,(p, A, p_2) \xrightarrow{c} q\,(p', B_1, p_1)\,(p_1, B_2, p_2) \in \Delta'$$

- Si $pA \xrightarrow{c} p' \in \Delta$ con $c \in (\Sigma \cup \{\epsilon\})$, entonces:

$$q\,(p, A, p') \xrightarrow{c} q \in \Delta'$$

Hipótesis de inducción (en el número de pasos n). Para todo $p, p' \in Q$, $A \in \Gamma$ y $w \in \Sigma^*$ se cumple que:

$$(pA, w) \vdash_{\mathcal{P}}^{n} (p', \epsilon) \quad \text{si, y solo si,} \quad (q\,(p, A, p')\,, w) \vdash_{\mathcal{P}'}^{n} (q, \epsilon)$$

donde $\vdash_{\mathcal{P}}^{n}$ es la relación de **siguiente-paso** de $\mathcal{P}$ n-veces.

Si demostramos esta hipótesis, habremos demostrado que $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{P}')$. ¿Por qué?

Caso base: $n = 1$. Para todo $p, p' \in Q$, y $A \in \Gamma$ se cumple que:

$$(pA, c) \vdash_{\mathcal{P}} (p', \epsilon) \quad \text{si, y solo si,} \quad (q\,(p, A, p')\,, c) \vdash_{\mathcal{P}'} (q, \epsilon)$$

para todo $c \in (\Sigma \cup \{\epsilon\})$.

Caso inductivo. Sin pérdida de generalidad, suponga que $pA \xrightarrow{a} p_1A_1A_2$ y $w = auv$, entonces

$$(pA, \underbrace{auv}_{w}) \vdash_{\mathcal{P}}^{n} (p', \epsilon) \text{ ssi } (pA, auv) \vdash_{\mathcal{P}} (p_1A_1A_2, uv) \vdash_{\mathcal{P}}^{i} (p_2A_2, v) \vdash_{\mathcal{P}}^{j} (p', \epsilon)$$

$$\begin{aligned}
&\text{ssi } (p_1A_1, u) \vdash_{\mathcal{P}}^{i} (p_2, \epsilon) \text{ y } \quad (p_2A_2, v) \vdash_{\mathcal{P}}^{j} (p', \epsilon) \\
&\text{ssi } (q\,(p_1, A_1, p_2)\,, u) \vdash_{\mathcal{P}'}^{i} (q, \epsilon) \text{ y } \quad (q\,(p_2, A_2, p')\,, v) \vdash_{\mathcal{P}'}^{j} (q, \epsilon) \\
&\text{ssi } (q\,(p, A, p')\,, auv) \vdash_{\mathcal{P}} (q\,(p_1, A_1, p_2)\,(p_2, A_2, q))\,, uv) \vdash_{\mathcal{P}}^{i+j} (q, \epsilon)
\end{aligned}$$

■

Paso 2. Sea $\mathcal{P} = (\{q\}, \Sigma, \Gamma, \Delta, q, \perp, \{q\})$ un PDA con **UN solo estado**. Construimos la gramática:

$$\mathcal{G} = (V, \Sigma, P, \perp)$$

- $V = \Gamma$.
- Si $qA \xrightarrow{\epsilon} q\alpha \in \Delta$ entonces $A \to \alpha \in P$
- Si $qA \xrightarrow{a} q\alpha \in \Delta$ entonces $A \to a\alpha \in P$

La demostración de este paso queda como ejercicio propuesto al lector.

5.3. Parsing: cómputo de First y Follow

Recordatorio. La **sintaxis** de un lenguaje es un conjunto de reglas que describen los programas válidos que tienen significado. Por otro lado, la **semántica** de un lenguaje define el significado de un programa correcto según la sintaxis.

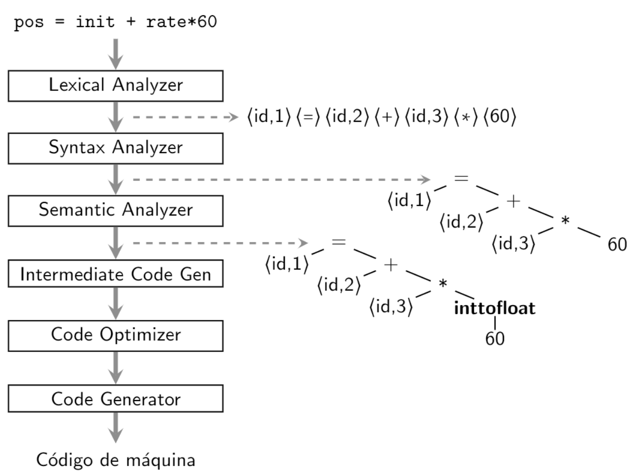


Figura 7: La estructura de un compilador

Lo que se busca es un proceso de **verificación de sintaxis** de un programa, y que entregue la estructura del mismo (árbol de parsing). Consta de tres etapas:

1. Análisis léxico (**Lexer**).
2. Análisis sintático (**Parser**).
3. Análisis semántico.

En una sección anterior vimos el **Lexer**. Ahora, veremos como hacer el **Parser**.

Informalmente: *"dado una secuencia de tokens w' y una gramática $\mathcal{G}$, construir un árbol de derivación (parsing) de $\mathcal{G}$ para w"*.

Con el **árbol de derivación** habremos verificado la sintaxis y obtenido la estructura.

Ejemplo 5.5: Parsing de gramática

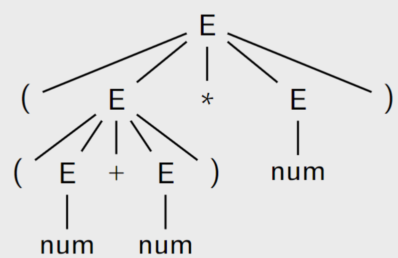
$$E \;\to\; (E+E) \mid (E*E) \mid \text{num}$$

Para un input $w = ((43+56)*27)$:

- Convertimos w en una secuencia de **tokens**:

$$w' = ((\text{num}+\text{num})*\text{num})$$

- Construimos un árbol de **parsing** para w':



Problema de parsing. Dado una palabra w y dado una gramática $\mathcal{G}$, generar un árbol de parsing $\mathcal{T}$ de $\mathcal{G}$ para w. ¿Ya sabemos resolver este problema? El algoritmo CKY nos permite hacer esto, pero:

- es impracticable para grandes inputs.
- múltiples pasadas sobre el input.

Deseamos hacer parsing en **tiempo lineal** en el tamaño del input. ¿Quién nos puede rescatar ante tal problema? Efectivamente, los autómatas apiladores.

Recordemos que, para una gramática $\mathcal{G} = (V, \Sigma, P, S)$ **podemos construir un PDA alternativo** $\mathcal{D}$ que acepta $\mathcal{L}(\mathcal{G})$:

$$\mathcal{D} = \Big(V \cup \Sigma \cup \{q_0, q_f\}, \Sigma, \Delta, q_0, \{q_f\}\Big)$$

La relación de transición Δ se define como:

$$\begin{array}{rcll} \Delta & = & \{(q_0, \epsilon, S \cdot q_f)\} & \cup \\ & & \{(X, \epsilon, \gamma) \mid X \to \gamma \in P\} & \cup \quad \textbf{(Expandir)} \\ & & \{(a, a, \epsilon) \mid a \in \Sigma\} & \quad\;\; \textbf{(Reducir)} \end{array}$$

Con esto, nos encontramos con otro **problema**: hay muchas alternativas para **expandir**. ¿Cómo elegir entonces la siguiente producción para expandir? Por ejemplo, si tenemos la regla $X \to \alpha \mid \beta$, ¿cómo elegir entre α o β?

Queremos elegir la **próxima producción** $X \to \gamma$ de tal manera que, si existe una derivación para el input, entonces $X \to \gamma$ **es parte de esa derivación**:

$$\text{si} \quad S \overset{\star}{\underset{\text{lm}}{\Rightarrow}} uX\gamma' \overset{\star}{\underset{\text{lm}}{\Rightarrow}} uv, \text{ entonces } \quad \gamma\gamma' \overset{\star}{\underset{\text{lm}}{\Rightarrow}} v$$

Necesitamos **mirar las siguientes letras en** v y ver si pueden ser producidas por α o β. Para esto, ocuparemos los conceptos de **first** y **follow**.

5.3.1. Prefijos

Definición. Sea Σ un alfabeto finito. Para un $k \geq 0$, se define

$$\boxed{\begin{aligned} \Sigma^{\leq k} &= \bigcup_{i=0}^{k} \Sigma^i \\ \Sigma_{\#}^{\leq k} &= \Sigma^{\leq k} \cup \left(\Sigma^{\leq k-1} \cdot \{\#\}\right) \end{aligned}}$$

Ejemplo 5.6

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

- $\Sigma^{\leq 2} = \{\epsilon, a, b, aa, ab, ba, bb\}$
- $\Sigma_{\#}^{\leq 2} = \{\epsilon, a, b, aa, ab, ba, bb\} \cup \{\#, a\#, b\#\}$

El símbolo # representará un EOF (End Of File), marcando el **fin de una palabra**.

Definición. Para una palabra $w = a_1 a_2 \ldots a_n \in \Sigma^*$ se define el k-**prefijo** de w como:

$$\boxed{w|_k = \begin{cases} a_1 \ldots a_n & \text{si } n \leq k \\ a_1 \ldots a_k & \text{si } k < n \end{cases}}$$

Definimos la k-**concatenación** $\odot_k$ entre strings $u, v \in \Sigma$ como:

$$\boxed{u \odot_k v = (u \cdot v)|_k}$$

Ejemplo 5.7

Sea $\Sigma = \{a, b\}$, entonces:

- $(abaa)|_2 = ab \qquad (ab)|_2 = ab \qquad (a)|_2 = a \qquad (\epsilon)|_2 = \epsilon$
- $a \odot_2 baa = (abaa)|_2 = ab$
- $bba \odot_2 a = (bbaa)|_2 = bb$
- $b \odot_2 \epsilon = (b)|_2 = b$

Extendemos estas operaciones para lenguajes $L, L_1, L_2 \subseteq \Sigma^*$ como:

$$\boxed{\begin{aligned} L|_k &= \{\, w|_k \mid w \in L\} \\ L_1 \odot_k L_2 &= \{w_1 \odot_k w_2 \mid w_1 \in L_1 \text{ y } w_2 \in L_2\} \end{aligned}}$$

Ejemplo 5.8

- $((ab)^*)|_3 = \{\epsilon, ab, aba\}$
- $(a)^* \odot_3 (ab)^* = \{\epsilon, a, aa, aaa, ab, aba, aab\}$

Podemos decir que los operadores $|_k$ y $\odot_k$ "miran" hasta un prefijo k.

Propiedades. Para todo $k \geq 1$ y $L_1, L_2, L_3 \subseteq \Sigma^*$:

1. $L_1 \odot_k (L_2 \odot_k L_3) = (L_1 \odot_k L_2) \odot_k L_3$
2. $L_1 \odot_k \{\epsilon\} = \{\epsilon\} \odot_k L_1 = L_1|_k$
3. $(L_1 L_2)|_k = L_1|_k \odot_k L_2|_k$
4. $L_1 \odot_k (L_2 \cup L_3) = (L_1 \odot_k L_2) \cup (L_1 \odot_k L_3)$

La demostración de estas propiedades queda como ejercicio propuesto al lector.

5.3.2. First y Follow

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto y $k \geq 1$.

Definición. Se define la función $\texttt{first}_k : (V \cup \Sigma)^* \to 2^{\Sigma^{\leq k}}$ tal que, para $\gamma \in (V \cup \Sigma)^*$:

$$\boxed{\texttt{first}_k(\gamma) = \{u|_k \mid \gamma \overset{*}{\Rightarrow} u\}}$$

Ejemplo 5.9

$$E \quad \to \quad (E+E) \mid (E*E) \mid n$$

- $\texttt{first}_1(E) = \{(, n\}$
- $\texttt{first}_2(E) = \{n, (n, ((\}$
- $\texttt{first}_3(E) = \{n, (n+, (n*, ((n, (((\}$

Definición. Se define la función $\texttt{follow}_k : V \to 2^{\Sigma_{\#}^{\leq k}}$ como:

$$\boxed{\texttt{follow}_k(X) = \{w \mid S \overset{*}{\Rightarrow} \alpha X \beta \text{ y } w \in \texttt{first}_k(\beta\#)\}}$$

Ejemplo 5.10

$$E \quad \to \quad (E+E) \mid (E*E) \mid n$$

- $\texttt{follow}_1(E) = \{\#, +, *,)\}$
- $\texttt{follow}_2(E) = \{\#,)\#,)),)+,)*, +(, *(, +n, *n\}$

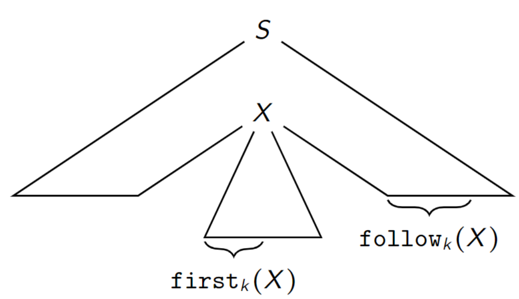


Figura 8: Representación de `first` y `follow`

5.3.3. Calcular First

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto y $k \geq 1$.

Proposición. Para $X_1, \ldots, X_n \in (V \cup \Sigma)$:

$$\boxed{\texttt{first}_k\left(X_1 \ldots X_n\right) = \texttt{first}_k\left(X_1\right) \odot_k \cdots \odot_k \texttt{first}_k\left(X_n\right)}$$

Demostración. Defina $\mathcal{L}(X) = \{w \mid X \stackrel{*}{\Rightarrow} w\}$ y $\mathcal{L}(\gamma) = \{w \mid \gamma \stackrel{*}{\Rightarrow} w\}$. Notar que $\texttt{first}_k(\gamma) = \mathcal{L}(\gamma)|_k$, por lo tanto, tenemos que

$$\begin{aligned}
\texttt{first}_k\left(X_1 \ldots X_n\right) &= \mathcal{L}\left(X_1 \ldots X_n\right)|_k \\
&= \left(\mathcal{L}\left(X_1\right) \cdot \mathcal{L}\left(X_2\right) \cdot \ldots \cdot \mathcal{L}\left(X_n\right)\right)|_k \\
&= \mathcal{L}\left(X_1\right)_k \odot_k \mathcal{L}\left(X_2\right)_k \odot_k \ldots \odot_k \mathcal{L}\left(X_n\right)|_k \\
&= \texttt{first}_k\left(X_1\right) \odot_k \cdots \odot_k \texttt{first}_k\left(X_n\right)
\end{aligned}$$

■

En particular, tenemos que:

$$\boxed{\texttt{first}_k(X) = \bigcup_{X \to X_1 \ldots X_n \in P} \texttt{first}_k(X_1) \odot_k \cdots \odot_k \texttt{first}_k(X_n)}$$

Definimos el siguiente **programa recursivo** para todo $X \in (V \cup \Sigma)$:

$$\begin{aligned}
\texttt{first}_k^0(X) &:= \bigcup_{X \to w \in P} w|_k \\
\texttt{first}_k^i(X) &:= \bigcup_{X \to X_1 \ldots X_n \in P} \texttt{first}_k^{i-1}\left(X_1\right) \odot_k \cdots \odot_k \texttt{first}_k^{i-1}\left(X_n\right)
\end{aligned}$$

Es fácil ver que:

- $\texttt{first}_k^{i-1}(X) \subseteq \texttt{first}_k^i(X)$ para todo $i > 1$.
- Como $\texttt{first}_k(X) \subseteq \Sigma^{\leq k}$, entonces para algún $i \leq k \cdot |\Sigma|^k \cdot |V|$ tendremos:

$$\texttt{first}_k^j(X) = \texttt{first}_k^{j+1}(X) \text{ para todo } j \geq i$$

Teorema 15

Sea i^* el menor número tal que $\texttt{first}_k^{i^*}(X) = \texttt{first}_k^{i^*+1}(X)$ para todo $X \in V$. Entonces, para todo $X \in V$:

$$\texttt{first}_k^{i^*}(X) = \texttt{first}_k(X)$$

La demostración del teorema anterior queda como ejercicio propuesto para el lector. Una idea para la dirección $\subseteq$, es demostrar por inducción que $\texttt{first}_k^i(X) \subseteq \texttt{first}_k(X)$. Para la dirección $\supseteq$, una idea es demostrar por inducción que si $X \stackrel{*}{\Rightarrow} w$, entonces $w|_k \in \texttt{first}_k^i(X)$ para algún i.

Algoritmo. A continuación se presenta un algoritmo para calcular $\texttt{first}_k$:

- **Input:** Gramática $\mathcal{G} = (V, \Sigma, P, S)$ y $k \geq 1$.
- **Output:** Todos los conjuntos $\texttt{first}_k$(X) para todo $X \in (V \cup \Sigma)$.

Function `CalcularFirst`$(\mathcal{G}, k)$**:**
- **foreach** $a \in \Sigma$ **do**
  - $\texttt{first}_k^0(a) := \{a\}$
- **foreach** $X \in V$ **do**
  - $\texttt{first}_k^0(X) := \cup_{X \to w \in P}\ w|_k$
- $i := 0$
- **repeat**
  - $i := i + 1$
  - **foreach** $a \in \Sigma$ **do**
    - $\texttt{first}_k^i(a) := \{a\}$
  - **foreach** $X \in V$ **do**
    - $\texttt{first}_k^i(X) := \displaystyle\bigcup_{X \to X_1 \dots X_n \in P} \texttt{first}_k^{i-1}(X_1) \odot_k \cdots \odot_k \texttt{first}_k^{i-1}(X_n)$
- **until** $\texttt{first}_k^i(X) = \texttt{first}_k^{i-1}(X)$ para todo $X \in (V \cup \Sigma)$
- **return** $\{\texttt{first}_k(X)\}_{X \in (V \cup \Sigma)}$

5.3.4. Calcular Follow

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto y $k \geq 1$. Si consideramos $X \neq S$:

$$\begin{aligned}
\texttt{follow}_k(X) &= \bigcup_{S \overset{*}{\Rightarrow} \alpha X \beta} \texttt{first}_k(\beta\#) \\
&= \bigcup_{S \overset{*}{\Rightarrow} \alpha Y \beta \Rightarrow \alpha\alpha' X \beta' \beta} \texttt{first}_k(\beta'\beta\#) \\
&= \bigcup_{Y \to \alpha' X \beta'} \bigcup_{S \overset{*}{\Rightarrow} \alpha Y \beta} \texttt{first}_k(\beta'\beta\#) \\
&= \bigcup_{Y \to \alpha' X \beta'} \bigcup_{S \overset{*}{\Rightarrow} \alpha Y \beta} \texttt{first}_k(\beta') \odot_k \texttt{first}_k(\beta\#) \\
&= \bigcup_{Y \to \alpha' X \beta'} \texttt{first}_k(\beta') \odot_k \bigcup_{S \overset{*}{\Rightarrow} \alpha Y \beta} \texttt{first}_k(\beta\#) \\
&= \bigcup_{Y \to \alpha' X \beta'} \texttt{first}_k(\beta') \odot_k \texttt{follow}_k(Y)
\end{aligned}$$

Si consideramos $X = S$:

$$\begin{aligned}
\texttt{follow}_k(S) &= \{\#\} \cup \bigcup_{S \overset{+}{\Rightarrow} \alpha S \beta} \texttt{first}_k(\beta\#) \\
&= \{\#\} \cup \bigcup_{S \overset{*}{\Rightarrow} \alpha Y \beta \Rightarrow \alpha\alpha' S \beta' \beta} \texttt{first}_k(\beta'\beta\#) \\
&= \{\#\} \cup \bigcup_{Y \to \alpha' S \beta'} \texttt{first}_k(\beta') \odot_k \texttt{follow}_k(Y)
\end{aligned}$$

Dado lo anterior, podemos definir el siguiente teorema.

Teorema 16

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto y $k \geq 1$. Entonces:

$$\text{Para } X \neq S: \qquad \texttt{follow}_k(X) = \bigcup_{Y \to \alpha X \beta} \texttt{first}_k(\beta) \odot_k \texttt{follow}_k(Y)$$

$$\text{Para } X = S: \qquad \texttt{follow}_k(S) = \{\#\} \cup \bigcup_{Y \to \alpha S \beta} \texttt{first}_k(\beta) \odot_k \texttt{follow}_k(Y)$$

Definimos el siguiente **programa recursivo** para todo $X \in V$:

$$\begin{array}{llll}
\text{Para } X \neq S: & \texttt{follow}_k^0(X) & := & \varnothing \\
\text{Para } X = S: & \texttt{follow}_k^0(S) & := & \{\#\} \\
\text{Para } X \neq S: & \texttt{follow}_k^i(X) & := & \displaystyle\bigcup_{Y \to \alpha X \beta} \texttt{first}_k(\beta) \odot_k \texttt{follow}_k^{i-1}(Y) \\
\text{Para } X = S: & \texttt{follow}_k^i(S) & := & \{\#\} \cup \displaystyle\bigcup_{Y \to \alpha S \beta} \texttt{first}_k(\beta) \odot_k \texttt{follow}_k^{i-1}(Y)
\end{array}$$

Similar al caso de $\texttt{first}_k$, es fácil ver que:

- $\texttt{follow}_k^{i-1}(X) \subseteq \texttt{follow}_k^i(X)$ para todo $i > 1$.
- Como $\texttt{follow}_k(X) \subseteq \Sigma^{\leq k}$, entonces para algún $i \leq k \cdot |\Sigma|^k \cdot |V|$:

$$\texttt{follow}_k^j(X) = \texttt{follow}_k^{j+1}(X) \quad \text{para todo } j \geq i$$

Teorema 17

Sea i^* el menor número tal que $\texttt{follow}_k^{i^*}(X) = \texttt{follow}_k^{i^*+1}(X)$ para todo $X \in V$. Entonces, para todo $X \in V$:

$$\texttt{follow}_k^{i^*}(X) = \texttt{follow}_k(X)$$

La demostración de este teorema se deja como ejercicio propuesto al lector.

Con todo lo anterior, podemos calcular $\texttt{follow}_k(X)$ con un algoritmo similar que $\texttt{first}_k(X)$. Respecto a la eficiencia de este tipo de algoritmos:

- Toman $\mathcal{O}(k \cdot |\Sigma|^k \cdot |V|)$ en el peor caso.
- Si $k = 1$, el número de repeticiones será $\mathcal{O}(|\Sigma| \cdot |V|)$ y el tiempo del algoritmo será polinomial en $|\mathcal{G}|$ en el peor caso. Incluso, se puede hacer en tiempo $\mathcal{O}(|V| \cdot |P|)$ en total.

5.4. Gramáticas LL

Volvamos a la idea de buscar un algoritmo que haga parsing en **tiempo lineal**. Para esto, contruíamos un autómata apilador alternativo $\mathcal{D}$ al cual le expandíamos sus producciones. ¿El problema? No sabemos como elegir que producciones expandir. Debido a lo anterior, introducimos los conceptos de `first` y `follow`. Así que, si tenemos una producción de la forma $X \to \alpha \mid \beta$, ¿cómo elegir entre α o β?

Estrategia (intuición). La idea es la siguiente:

1. Mirar k símbolos del resto del input v (k-**lookahead**).
2. Usar $v|_k$ y decidir cuál regla $X \to \gamma$ elegimos para expandir.

La caracterización de las gramáticas que cumplen las propiedades anteriores se denominan **Gramáticas** LL(k), donde

- **Primera** L: leer el input de izquierda a derecha (**L**eft-right).
- **Segunda** L: producir una derivación por la izquierda (**L**eftmost).
- **Parámetro** k: el número de letras en adelante que utiliza (**lookahead**).

5.4.1. Definición Gramáticas LL

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto y $k \geq 1$.

Definición. Decimos que $\mathcal{G}$ es una gramática LL(k) si para todas las derivaciones:

- $S \underset{\text{lm}}{\overset{*}{\Rightarrow}} uY\beta \underset{\text{lm}}{\Rightarrow} u\gamma_1\beta \underset{\text{lm}}{\overset{*}{\Rightarrow}} uv_1$
- $S \underset{\text{lm}}{\overset{*}{\Rightarrow}} uY\beta \underset{\text{lm}}{\Rightarrow} u\gamma_2\beta \underset{\text{lm}}{\overset{*}{\Rightarrow}} uv_2$ y
- $v_1|_k = v_2|_k$

entonces se cumple que $\gamma_1 = \gamma_2$.

Notar que la elección de $Y \to \gamma$ depende de Y, $v|_k$ y u.

Ejemplo 5.11: Gramáticas LL(1)

$$\mathcal{G}_1: \quad S \rightarrow (S) \mid n$$

$$\begin{array}{lllll} S \underset{\text{lm}}{\overset{\star}{\Rightarrow}} & \overbrace{(\cdots(}^{u}\overbrace{S}^{Y}\overbrace{)\cdots)}^{\beta} & \underset{\text{lm}}{\Rightarrow} (\cdots(\gamma_1)\cdots) & \underset{\text{lm}}{\overset{\star}{\Rightarrow}} & \overbrace{(\cdots(}^{u}\overbrace{v_1')\cdots)}^{v_1} \\ & & \underset{\text{lm}}{\Rightarrow} (\cdots(\gamma_2)\cdots) & \underset{\text{lm}}{\overset{\star}{\Rightarrow}} & \underbrace{(\cdots(}_{u}\underbrace{v_2')\cdots)}_{v_2} \end{array}$$

- Si $v_1|_1 = v_2|_1 = n$, entonces $\gamma_1 = \gamma_2 = n$.
- Si $v_1|_1 = v_2|_1 = ($, entonces $\gamma_1 = \gamma_2 = (S)$.

En ambos casos, tenemos que $\gamma_1 = \gamma_2$ y $\mathcal{G}_1$ es una gramática LL(1).

$$\begin{array}{llll} \mathcal{G}_2: & S & \rightarrow & (X) \mid n \\ & X & \rightarrow & SX \mid \epsilon \end{array}$$

$$\begin{array}{lllll} S \underset{\text{lm}}{\overset{\star}{\Rightarrow}} & uX\beta & \underset{\text{lm}}{\Rightarrow} u\gamma_1\beta & \underset{\text{lm}}{\overset{\star}{\Rightarrow}} & uv_1 \\ & & \underset{\text{lm}}{\Rightarrow} u\gamma_2\beta & \underset{\text{lm}}{\overset{\star}{\Rightarrow}} & uv_2 \end{array}$$

- Si $v_1|_1 = v_2|_1 = ($ o 'n', entonces $\gamma_1 = \gamma_2 = SX$.
- Si $v_1|_1 = v_2|_1 =)$, entonces $\gamma_1 = \gamma_2 = \epsilon$.

Por lo tanto, tenemos que $\gamma_1 = \gamma_2$ y $\mathcal{G}_2$ es **también** una gramática LL(1).

Ejemplo 5.12: Gramática NO LL(1) pero si LL(2)

$$\begin{array}{llcl} \mathcal{G}_3: & S & \rightarrow & (X) \mid n+S \mid n \\ & X & \rightarrow & SX \mid \epsilon \end{array}$$

$$S \overset{\star}{\underset{\text{lm}}{\Rightarrow}} \overbrace{(}^{u}\,\overbrace{S}^{Y}\,\overbrace{X)}^{\beta} \begin{array}{l} \underset{\text{lm}}{\Rightarrow} (\,\overbrace{n+S}^{\gamma_1}\;X) \overset{\star}{\underset{\text{lm}}{\Rightarrow}} \overbrace{(}^{u}\,\overbrace{n+n)}^{v_1} \\ \underset{\text{lm}}{\Rightarrow} (\,\underbrace{n}_{\gamma_2}\;X) \overset{\star}{\underset{\text{lm}}{\Rightarrow}} \underbrace{(}_{u}\,\underbrace{n)}_{v_2} \end{array} \qquad S \overset{\star}{\underset{\text{lm}}{\Rightarrow}} uS\beta \begin{array}{l} \underset{\text{lm}}{\Rightarrow} u\gamma_1\beta \overset{\star}{\underset{\text{lm}}{\Rightarrow}} uv_1 \\ \underset{\text{lm}}{\Rightarrow} u\gamma_2\beta \overset{\star}{\underset{\text{lm}}{\Rightarrow}} uv_2 \end{array}$$

Como $v_1|_1 = v_2|_1 = n$ pero $\gamma_1 \neq \gamma_2$, entonces $\mathcal{G}_3$ NO es una gramática LL(1).

- ♦ Si $v_1|_2 = v_2|_2 = n+$, entonces $\gamma_1 = \gamma_2 = n + S$.
- ♦ Si $v_1|_1 = v_2|_1 = na$, con $a \neq +$, entonces $\gamma_1 = \gamma_2 = n$.

Por lo tanto, tenemos que $\gamma_1 = \gamma_2$ y entonces $\mathcal{G}_3$ es LL(2).

Ejemplo 5.13: Gramática NO LL(k)

$$\begin{array}{llcl} \mathcal{G}_4: & S & \rightarrow & (X) \mid (X)\hat{\ }e \mid n+S \mid n \\ & X & \rightarrow & SX \mid \epsilon \end{array}$$

$$S \overset{\star}{\underset{\text{lm}}{\Rightarrow}} \overbrace{(}^{u}\,\overbrace{S}^{Y}\,\overbrace{X)}^{\beta} \begin{array}{l} \underset{\text{lm}}{\Rightarrow} (\,\overbrace{(S)\hat{\ }e}^{\gamma_1}\;X) \overset{\star}{\underset{\text{lm}}{\Rightarrow}} \overbrace{(}^{u}\,\overbrace{(\overset{k}{\cdots}(n)\overset{k}{\cdots})\hat{\ }e)}^{v_1} \\ \underset{\text{lm}}{\Rightarrow} (\,\underbrace{(S)}_{\gamma_2}\;X) \overset{\star}{\underset{\text{lm}}{\Rightarrow}} \underbrace{(}_{u}\,\underbrace{(\overset{k}{\cdots}(n)\overset{k}{\cdots}))}_{v_2} \end{array}$$

Como $v_1|_k = v_2|_k = (\overset{k}{\cdots}($ pero $\gamma_1 \neq \gamma_2$, entonces $\mathcal{G}_4$ NO es una gramática LL(k) para todo k.

Ejemplo 5.14: Gramática NO LL(k) transformada en LL(2)

La gramática $\mathcal{G}_4$ del ejemplo anterior se puede transformar para que sea LL(2) de la siguiente manera:

$$\begin{array}{llcl} \mathcal{G}_4': & S & \rightarrow & (XY \mid n+S \mid n \\ & X & \rightarrow & SX \mid \epsilon \\ & Y & \rightarrow &) \mid)\hat{\ }e \end{array}$$

Queda como ejecicio para el lector demostrar que $\mathcal{G}_4'$ es LL(2).

Ejemplo 5.15: Lenguaje NO LL(k)

$$\mathcal{G}_5: \begin{array}{rcl} S & \rightarrow & X \mid Y \\ X & \rightarrow & aXb \mid 0 \\ Y & \rightarrow & aYbb \mid 1 \end{array} \qquad \begin{array}{rcl} \mathcal{L}(\mathcal{G}_5) & = & \{a^n 0 b^n \mid n \geq 0\} \cup \\ & & \{a^n 1 b^{2n} \mid n \geq 0\} \end{array}$$

$$\begin{array}{ccccccc} S & \underset{\text{lm}}{\overset{\star}{\Rightarrow}} & S & \underset{\text{lm}}{\Rightarrow} & X & \underset{\text{lm}}{\overset{\star}{\Rightarrow}} & a^k 0 b^k \\ & & & \underset{\text{lm}}{\Rightarrow} & Y & \underset{\text{lm}}{\overset{\star}{\Rightarrow}} & a^k 1 b^{2k} \end{array}$$

Para todo $k \geq 1$, se tiene que $\mathcal{G}_5$ NO es una gramática LL(k).

Es posible demostrar que, para toda gramática $\mathcal{G}$ con $\mathcal{L}(\mathcal{G}_5) = \mathcal{L}(\mathcal{G})$, $\mathcal{G}$ NO es una gramática LL(k) para todo $k \geq 1$.

5.4.2. Caracterización LL

Para esta parte es importante manejar las definiciones de prefijos vistas en la sección 5.3.1.

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto **reducida** y $k \geq 1$. En base a esto definimos el siguiente teorema:

Teorema 18

$\mathcal{G}$ es una gramática LL(k) si, y sólo si, para todas dos reglas distintas $Y \rightarrow \gamma_1, Y \rightarrow \gamma_2 \in P$ y para todo $S \underset{\text{lm}}{\overset{*}{\Rightarrow}} uY\beta$, se tiene que:

$$\texttt{first}_k(\gamma_1\beta) \cap \texttt{first}_k(\gamma_2\beta) = \varnothing$$

Demostración. ($\Rightarrow$) Por contrapositivo, supongamos que $v \in \texttt{first}_k(\gamma_1\beta) \cap \texttt{first}_k(\gamma_2\beta)$. Como $\mathcal{G}$ es reducida (sin variables inútiles), entonces

$$\begin{array}{ccccccc} S & \underset{\text{lm}}{\overset{*}{\Rightarrow}} & uY\beta & \underset{\text{lm}}{\Rightarrow} & u\gamma_1\beta & \underset{\text{lm}}{\overset{*}{\Rightarrow}} & uvv_1 \\ & & & \underset{\text{lm}}{\Rightarrow} & u\gamma_2\beta & \underset{\text{lm}}{\overset{*}{\Rightarrow}} & uvv_2 \end{array}$$

para algún $v_1, v_2 \in \Sigma^*$. Como $\gamma_1 \neq \gamma_2$, entonces $\mathcal{G}$ NO es LL(k).

($\Leftarrow$) Por contrapositivo (de nuevo), supongamos que $\mathcal{G}$ no es LL(k). Como $\mathcal{G}$ no es LL(k), entonces tenemos derivaciones de la forma:

$$\begin{array}{ccccccc} S & \underset{\text{lm}}{\overset{*}{\Rightarrow}} & uY\beta & \underset{\text{lm}}{\Rightarrow} & u\gamma_1\beta & \underset{\text{lm}}{\overset{*}{\Rightarrow}} & uv_1 \\ & & & \underset{\text{lm}}{\Rightarrow} & u\gamma_2\beta & \underset{\text{lm}}{\overset{*}{\Rightarrow}} & uv_2 \end{array}$$

Vemos que $v_1|_k = v_2|_k = v$, pero $\gamma_1 \neq \gamma_2$. Por lo tanto, $v \in \texttt{first}_k(\gamma_1\beta) \cap \texttt{first}_k(\gamma_2\beta)$. ∎

¿Cómo usamos la caracterización del teorema para demostrar que una gramática es LL(k)? Buscaremos condiciones más simples para verificar si una gramática es LL(k).

Definición. $\mathcal{G}$ es una gramática LL(k) **fuerte** si para todas dos reglas distintas $Y \rightarrow \gamma_1, Y \rightarrow \gamma_2 \in P$ se tiene que:

$$\boxed{\texttt{first}_k(\gamma_1) \odot_k \texttt{follow}_k(Y) \cap \texttt{first}_k(\gamma_2) \odot_k \texttt{follow}_k(Y) = \varnothing}$$

Ejemplo 5.16: Si $\mathcal{G}$ es LL(k) fuerte, entonces $\mathcal{G}$ es LL(k)

Una gramática $\mathcal{G}$ que sea LL(k) fuerte siempre es LL(k), ya que si definimos dos conjuntos dados por el teorema de LL(k) (F_1) y la definición de LL(k) fuerte (F_2), dados por:

$$F_1 = \texttt{first}_k(\gamma_1\beta) \ \cap \ \texttt{first}_k(\gamma_2\beta) = \texttt{first}_k(\gamma_1) \odot_k \texttt{first}_k(\beta) \ \cap \ \texttt{first}_k(\gamma_2) \odot_k \texttt{first}_k(\beta)$$
$$F_2 = \texttt{first}_k(\gamma_1) \odot_k \texttt{first}_k(Y) \ \cap \ \texttt{first}_k(\gamma_2) \odot_k \texttt{first}_k(Y) = \varnothing$$

Entonces, tenemos que $F_1 \subseteq F_2$.

Ejemplo 5.17: Si $\mathcal{G}$ es LL(k), ¿es LL(k) fuerte?

La respuesta directa es que no. Con un contrajemplo, tomemos la gramática $\mathcal{G}$ definida por

$$\begin{aligned} \mathcal{G}: \quad & S \to aXaa \mid bXba \\ & X \to b \mid \epsilon \end{aligned}$$

Recordatorio: $\mathcal{G}$ es LL(k) si para todas dos reglas distintas $Y \to \gamma_1, Y \to \gamma_2 \in P$ y para todo $S \underset{\text{lm}}{\overset{*}{\Rightarrow}} uY\beta$, se tiene que

$$\texttt{first}_k(\gamma_1\beta) \ \cap \ \texttt{first}_k(\gamma_2\beta) = \varnothing$$

- Si $S \underset{\text{lm}}{\overset{*}{\Rightarrow}} aXaa$, entonces $\texttt{first}_2(baa) \ \cap \ \texttt{first}_2(aa) = \varnothing$.
- Si $S \underset{\text{lm}}{\overset{*}{\Rightarrow}} bXba$, entonces $\texttt{first}_2(baa) \ \cap \ \texttt{first}_2(ba) = \varnothing$

Por lo tanto, $\mathcal{G}$ es LL(2).

Recordatorio: $\mathcal{G}$ es una gramática LL(k) fuerte si para todas dos reglas distintas $Y \to \gamma_1, Y \to \gamma_2 \in P$ se tiene que:

$$\texttt{first}_k(\gamma_1) \odot_k \texttt{follow}_k(Y) \ \cap \ \texttt{first}_k(\gamma_2) \odot_k \texttt{follow}_k(Y) = \varnothing$$

Si vemos $X \to b$ y $X \to \epsilon$:

$$\begin{aligned} \texttt{first}_2(b) \odot_2 \texttt{follow}_2(X) \ \cap \ \texttt{first}_2(\epsilon) \odot_2 \texttt{follow}_2(X) & \\ = \{b\} \odot_2 \{aa, ba\} \ \cap \ \{\epsilon\} \odot_2 \{aa, ba\} & \\ = \{ba, bb\} \ \cap \ \{aa, ba\} & \\ = \{ba\} & \qquad \text{y por ende } \mathcal{G} \text{ no es LL(2) fuerte.} \end{aligned}$$

¿Qué pasa con el caso LL(1)? Supongamos que $\mathcal{G}$ es LL(1) y $Y \to \gamma_1, Y \to \gamma_2 \in P$ son reglas distintas.

1. Si $\epsilon \notin \texttt{first}_1(\gamma_1)$ y $\epsilon \notin \texttt{first}_1(\gamma_2)$, entonces, por la caracterización de LL(1):

$$\begin{aligned} \varnothing &= \texttt{first}_1(\gamma_1\beta) \ \cap \ \texttt{first}_1(\gamma_2\beta) \\ &= \texttt{first}_1(\gamma_1) \ \cap \ \texttt{first}_1(\gamma_2) \\ &= \texttt{first}_1(\gamma_1) \odot_1 \texttt{follow}_1(Y) \ \cap \ \texttt{first}_1(\gamma_2) \odot_1 \texttt{follow}_1(Y) \end{aligned}$$

2. Si $\epsilon \in \texttt{first}_1(\gamma_1)$ y $\epsilon \notin \texttt{first}_1(\gamma_2)$, entonces, por la caracterización de LL(1):

$$\begin{aligned} \varnothing &= \texttt{first}_1(\gamma_1\beta) \ \cap \ \texttt{first}_1(\gamma_2\beta) \\ &= \texttt{first}_1(\gamma_1\beta) \ \cap \ \texttt{first}_1(\gamma_2) \\ &= \texttt{first}_1(\gamma_1\beta) \ \cap \ \texttt{first}_1(\gamma_2\beta') \end{aligned}$$

para todo $\beta' \in (V \cup \Sigma)^*$. Por lo tanto:

$$\begin{aligned} & \texttt{first}_1(\gamma_1) \odot_1 \texttt{follow}_1(Y) \ \cap \ \texttt{first}_1(\gamma_2) \odot_1 \texttt{follow}_1(Y) \\ = & \bigcup_{S \underset{\text{lm}}{\overset{*}{\Rightarrow}} uY\beta} \texttt{first}_1(\gamma_1\beta) \ \cap \bigcup_{S \underset{\text{lm}}{\overset{*}{\Rightarrow}} uY\beta'} \texttt{first}_1(\gamma_2\beta') = \varnothing \end{aligned}$$

Por lo tanto, establecemos el siguiente teorema.

Teorema 19

Una gramática $\mathcal{G}$ es LL(1) si, y sólo si, $\mathcal{G}$ es LL(1) ***fuerte***, *esto es, para todas dos reglas distintas $Y \to \gamma_1, Y \to \gamma_2 \in P$:*

$$\texttt{first}_1(\gamma_1) \odot_1 \texttt{follow}_1(Y) \ \cap \ \texttt{first}_1(\gamma_2) \odot_1 \texttt{follow}_1(Y) = \varnothing$$

La condición del teorema anterior se puede verificar en **tiempo polinomial** en $\mathcal{G}$.

5.5. Parsing con gramáticas LL(k)

5.5.1. Algunas consideraciones

Considere la siguiente gramática $\mathcal{G}$:

$$\begin{aligned} S &\to Xa \mid Xb \\ X &\to c \end{aligned}$$

¿Es esta gramática del tipo LL(1)? Podemos ver que $\texttt{first}_1(\gamma_1\beta) = \{c\}$ y $\texttt{first}_1(\gamma_2\beta) = \{c\}$, con $\gamma_1 = Xa$, $\gamma_2 = Xb$ y $\beta = \epsilon$. Por lo tanto su intersección no es vacía y entonces $\mathcal{G}$ no es LL(1). ¿Podemos establecer una solución para este problema?

Factorización. En general, si tenemos una regla:

$$X \to \gamma\alpha_1 \mid \gamma\alpha_2$$

siempre podemos "**factorizar**" la regla manteniendo la semántica, como:

$$\begin{aligned} X &\to \gamma X' \\ X' &\to \alpha_1 \mid \alpha_2 \end{aligned}$$

Considere ahora la siguiente gramática $\mathcal{G}$:

$$E \to E * E \mid n$$

¿Es esta gramática del tipo LL(1)? ¿LL(k)? Pues no es ninguna. El problema con esta gramática es su recursividad, en específico, por la izquierda.

Definición. Una gramática $\mathcal{G}$ se dice **recursiva por la izquirrda** si existe $X \in V$ tal que:

$$\boxed{X \overset{+}{\Rightarrow} X\gamma \quad \text{para algún } \gamma \in (V \cup \Sigma)^*}$$

Teorema 20

Si $\mathcal{G} = (V, \Sigma, P, S)$ es una gramática reducida y recursiva por la izquierda, entonces $\mathcal{G}$ NO es LL(k) para todo $k \geq 1$.

Demostración. Por simplicidad, suponga que $X \to X\beta \in P$ y $X \to w \in P$.
Como $\mathcal{G}$ es reducida, entonces existe una derivación $S \overset{*}{\underset{\text{lm}}{\Rightarrow}} uX\gamma$:

$$S \overset{*}{\underset{\text{lm}}{\Rightarrow}} uX\gamma \underset{\text{lm}}{\Rightarrow} \overset{n\text{-veces}}{\cdots} \underset{\text{lm}}{\Rightarrow} uX\beta^n\gamma$$

Por **contradicción**, suponga que $\mathcal{G}$ es LL(k). Por lo tanto:

$$\texttt{first}_k(X\beta^{n+1}\gamma) \cap \texttt{first}_k(w\beta^n\gamma) = \varnothing$$

Suponga que $\beta \overset{*}{\Rightarrow} v \in \Sigma^*$ y $\gamma \overset{*}{\Rightarrow} v' \in \Sigma^*$. Con $n = k$, tendremos que

$$(wv^kv')|_k \quad \in \quad \texttt{first}_k(X\beta^{k+1}\gamma) \cap \texttt{first}_k(w\beta^k\gamma) \quad \to\leftarrow \quad \text{(¡contradicción! el conjunto no es vacío)}$$

■

Hablemos de recursión **inmediata** por la izquierda. Suponga que existe $X \in V$ tal que:

$$X \to X\alpha_1 \mid \cdots \mid X\alpha_m \mid \beta_1 \mid \cdots \mid \beta_n$$

¿Cómo podemos **eliminar** la recursión inmediata por la izquierda? Consideramos la misma gramática pero **cambiando** las reglas de X por:

$$X \to \beta_1 X' \mid \cdots \mid \beta_n X'$$
$$X' \to \alpha_1 X' \mid \cdots \mid \alpha_m X' \mid \epsilon$$

Ejemplo 5.18: Eliminando recursión inmediata

$$\begin{array}{rcl} S & \to & Xa \mid b \\ X & \to & Xc \mid d \end{array} \qquad \Bigg| \qquad \begin{array}{rcl} S & \to & Xa \mid b \\ X & \to & dX' \\ X' & \to & cX' \mid \epsilon \end{array}$$

Teorema 21

Sea $\mathcal{G}$ una gramática tal que existe $X \in V$:

$$X \to X\alpha_1 \mid \cdots \mid X\alpha_m \mid \beta_1 \mid \cdots \mid \beta_n$$

Sea $\mathcal{G}'$ la misma gramática $\mathcal{G}$ pero cambiando las reglas de X por:

$$X \to \beta_1 X' \mid \cdots \mid \beta_n X'$$
$$X' \to \alpha_1 X' \mid \cdots \mid \alpha_m X' \mid \epsilon$$

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

Demostración. Una derivación por la izquierda de X en $\mathcal{G}$:

$$X \underset{\text{lm}}{\Rightarrow} X\alpha_{i_1} \underset{\text{lm}}{\Rightarrow} X\alpha_{i_2}\alpha_{i_1} \underset{\text{lm}}{\Rightarrow} \cdots \underset{\text{lm}}{\Rightarrow} X\alpha_{i_p}\alpha_{i_{p-1}}\cdots\alpha_{i_1} \underset{\text{lm}}{\Rightarrow} \beta_j\alpha_{i_p}\alpha_{i_{p-1}}\cdots\alpha_{i_1}$$

Una derivación por la derecha de X en $\mathcal{G}'$ **equivalente**:

$$X \underset{\text{rm}}{\Rightarrow} \beta_j X' \underset{\text{rm}}{\Rightarrow} \beta_j\alpha_{i_p}X' \underset{\text{rm}}{\Rightarrow} \cdots \underset{\text{rm}}{\Rightarrow} \beta_j\alpha_{i_p}\cdots\alpha_{i_2}\alpha_{i_1}X' \underset{\text{rm}}{\Rightarrow} \beta_j\alpha_{i_p}\alpha_{i_{p-1}}\cdots\alpha_{i_1}$$

■

Ahora, ¿qué pasa si la recursión por la izquierda es **no-inmediata**? Considere la siguiente gramática **recursiva por la izquierda**:

$$\begin{aligned} S &\to Xa \mid b \\ X &\to Yc \\ Y &\to Xd \mid e \end{aligned}$$

¿Cómo eliminamos la recursión por la izquierda no-inmediata?

Estrategia. Dado $V = \{X_1, \ldots, X_n\}$, removemos la recursión inductivamente en n, tal que, en cada paso i de la inducción, se cumplira que para todo $i, j \leq n$:

$$\text{si } X_i \to X_j\alpha, \text{ entonces } i < j$$

input : Gramática $\mathcal{G} = (V, \Sigma, P, S)$ y $V = \{X_1, \ldots, X_n\}$

output: Gramática $\mathcal{G}$ sin recursión por la izquierda

Function `EliminarRecursion(`$\mathcal{G}$`)`:

```
P' := P
for i = 1 to n do
    for j = 1 to i − 1 do
        foreach X_i → X_j γ ∈ P' do
            foreach X_j → α ∈ P' do
                P' := P' ∪ {X_i → αγ}
            P' := P' − {X_i → X_j γ}
    Remover recursión inmediata para X_i en P' (si existe)
V' := {X_1, ..., X_n} ∪ {X'_1, ..., X'_n}
return (V', Σ, P', S)
```

Queda como ejercicio propuesto al lector demostrar la correctitud del algoritmo.

Ejemplo 5.19: Eliminando recursión

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to (E) \mid n \end{aligned}$$

Eliminando la **recursión inmediata** de E:

$$\begin{aligned} E &\to TE' \\ E' &\to +TE' \mid \epsilon \\ T &\to T * F \mid F \\ F &\to (E) \mid n \end{aligned}$$

Eliminando la **recusión inmediata** de T:

$$\begin{aligned} E &\to TE' \\ E' &\to +TE' \mid \epsilon \\ T &\to FT' \\ T' &\to *FT' \mid \epsilon \\ F &\to (E) \mid n \end{aligned}$$

Conclusión. Es **posible eliminar** la recursividad por la izquierda, pero esto **NO asegura** que el resultado sea una gramática LL(k) para algún k.

5.5.2. Parsing de LL(k)

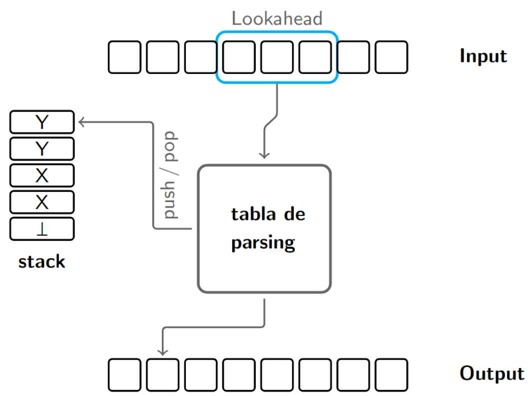


Figura 9: Máquina de parsing

Definición. Sea Σ un alfabeto finito. Se definen los siguientes conjuntos de palabras:

- $\dot{\Sigma} = \Sigma^* \times \Sigma^*$
- $\dot{\Sigma}^{\leq k} = \{(u,v) \in \dot{\Sigma} \mid |uv| \leq k\}$
- $\dot{\Sigma}^{\leq k}_{\#} = \{(u,v) \in \dot{\Sigma} \mid |uv| \leq k\} \cup \{(u, v\#) \mid (u,v) \in \dot{\Sigma} \mid |uv| < k\}$

Notación. En vez de usar $(u,v) \in \dot{\Sigma}^{\leq k}_{\#}$, escribiremos $u.v \in \dot{\Sigma}^{\leq k}_{\#}$. El par $\epsilon.\epsilon$ lo denotaremos solamente por ϵ.

Definición. Un transductor apilador con k-lookahead (k-PDT) es una tupla:

$$\boxed{\mathcal{T} = (Q, \Sigma, \Omega, \Delta, q_0, F)}$$

- Q es un conjunto finito de estados.
- Σ es el alfabeto de input.
- Ω es el alfabeto de output.
- $\Delta \subseteq Q^+ \times \dot{\Sigma}^{\leq k}_{\#} \times (\Omega \cup \{\epsilon\}) \times Q^*$ es la relación de transición.
- $q_0 \in Q$ es un conjunto de estados iniciales.
- $F \subseteq Q$ es el conjunto de estados finales.

Definición. Una **configuración** de $\mathcal{T}$ es una tupla:

$$\boxed{(q_1 \dots q_k, w, o) \in \left(Q^+, \Sigma^* \cdot \{\#\}, \Omega^*\right)}$$

- ♦ $q_1 \dots q_k$ es el contenido del stack con q_1 el tope del stack.
- ♦ w es el contenido del input.
- ♦ o es el contenido del output.

Decimos que una configuración:

- ♦ $(q_0, w\#, \epsilon)$ es **inicial** y
- ♦ $(q_f, \#, o)$ es **final** si $q_f \in F$.

Definición. Se define la relación $\vdash_{\mathcal{T}}$ de **siguiente-paso** entre configuraciones de $\mathcal{T}$:

$$\boxed{(\gamma_1, w_1, o_1) \ \vdash_{\mathcal{T}} \ (\gamma_2, w_2, o_2)}$$

si, y sólo si, existe $(\alpha, u.v, a, \beta) \in \Delta$, $\gamma \in \Gamma^*$ y $w \in \Sigma^* \cdot \{\#\}$ tal que:

- ♦ **Stack:** $\gamma_1 = \alpha \cdot \gamma$ y $\gamma_2 = \beta \cdot \gamma$
- ♦ **Look-ahead:** $w_1 = u \cdot v \cdot w$ y $w_2 = v \cdot w$
- ♦ **Output:** $o_2 = o_1 \cdot a$

Se define $\vdash_{\mathcal{T}}^*$ como la clausura **refleja** y **transitiva** de $\vdash_{\mathcal{T}}$.

Definición. $\mathcal{T}$ **entrega** o con **input** w si existe una configuración inicial $(q_0, w \cdot \#, \epsilon)$ y una configuración final $(q_f, \#, o)$ tal que:

$$(q_0, w \cdot \#, \epsilon) \vdash_{\mathcal{T}}^* (q_f, \#, o)$$

Se define la función $[\![\mathcal{T}]\!] : \Sigma^* \to 2^{\Omega^*}$:

$$\boxed{[\![\mathcal{T}]\!](w) = \{o \in \Omega^* \mid \mathcal{T} \text{ entrega } o \text{ con input } w\}}$$

Definición. $\mathcal{T}$ es **determinista** si para todo $(\alpha_1, u_1.v_1, a_1, \beta_1), (\alpha_2, u_2.v_2, a_2, \beta_2) \in \Delta$ con $(\alpha_1, u_1.v_1, a_1, \beta_1) \neq (\alpha_2, u_2.v_2, a_2, \beta_2)$ se cumple que

$$\boxed{\alpha_1 \text{ NO es prefijo de } \alpha_2 \quad \text{o} \quad u_1 v_1 \text{ NO es prefijo de } u_2 v_2.}$$

*"Para cualquier configuración (γ, w, o) existe **a lo más** una configuración (γ', w', o) tal que $(\gamma, w, o) \vdash_{\mathcal{T}}^* (\gamma', w', o')$"*

La **ventaja** de un k-PDT determinista es que nos aseguramos de que siempre obtenemos un solo output para cada input (el no-determinismo nos podría generar muchos outputs distintos).

Construcción del parser. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática LL(k) fuerte. Se define el k-PDT para $\mathcal{G}$:

$$\boxed{\mathcal{T}[\mathcal{G}] = \Big(V \cup \Sigma \cup \{q_0, q_f\}, \Sigma, \underbrace{P}_{\Omega}, \Delta, q_0, \{q_f\}\Big)}$$

La relación de transición Δ de define como:

Inicio: $(q_0, \epsilon., \epsilon, S \cdot q_f)$

Reducir: $(a, a., \epsilon, \epsilon)$ para cada $a \in \Sigma$

Expandir: $(X, .u, p, \gamma)$

para cada $p := (X \to \gamma) \in P$ tal que $u \in \mathtt{first}_k(\gamma) \odot_k \mathtt{follow}_k(X)$

Propiedades. $\mathcal{T}[\mathcal{G}]$ tiene las siguientes propiedades:

1. $\mathcal{T}[\mathcal{G}]$ es un k-PDT **determinista** si, y sólo si, $\mathcal{G}$ es LLk fuerte.
2. Si $w \notin \mathcal{L}(\mathcal{G})$ entonces $[\![\mathcal{T}]\!](w) = \varnothing$.
3. Si $w \in \mathcal{L}(\mathcal{G})$ entonces $[\![\mathcal{T}]\!](w) = \{r_1 \dots r_m\}$ es una derivación por la izquierda de $\mathcal{G}$ sobre w.

Algoritmo. Para una gramática LL(k) $\mathcal{G}$ y una palabra $w \in \Sigma^*$:

1. Construya el k-PDT determinista $\mathcal{T}[\mathcal{G}]$ a partir de $\mathcal{G}$.
2. Ejecute $\mathcal{T}[[G]]$ sobre w.

Como $\mathcal{T}[\mathcal{G}]$ es determinista, entonces el algoritmo toma **tiempo lineal** en w.

Tabla predictiva para LL(k) fuerte. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática LL(k) fuerte. Para cada $u \in \Sigma^k \cup \Sigma^{<k} \cdot \{\#\}$, se define $M[X, u] \in (V \cup \Sigma)^* \cup \{\mathtt{ERROR}\}$:

$$\boxed{M[X,u] = \begin{cases} \gamma & \text{si } X \to \gamma \in P \text{ y } u \in \mathtt{first}_k(\gamma) \odot_k \mathtt{follow}_k(X) \\ \mathtt{ERROR} & \text{en otro caso.} \end{cases}}$$

El computo de la tabla predictiva puede tomar **tiempo exponencial** en $|\mathcal{G}|$ y k.

Caso especial: tabla predictiva para LL(1). Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática LL(1) fuerte. Para cada $a \in \Sigma \cup \{\#\}$, se define $M[x, a] \in (V \cup \Sigma)^* \cup \{\mathtt{ERROR}\}$:

$$\boxed{M[X,a] = \begin{cases} \gamma & \text{si } X \to \gamma \in P \text{ y } a \in \mathtt{first}_1(\gamma) \\ \gamma & \text{si } X \to \gamma \in P, \epsilon \in \mathtt{first}_1(\gamma) \text{ y } a \in \mathtt{follow}_1(X) \\ \text{ERROR} & \text{en otro caso.} \end{cases}}$$

Este cálculo se puede hacer en tiempo $\mathcal{O}(|V| \cdot |P|)$.

Ejemplo 5.20: Tabla predictiva

$$
\begin{array}{lcl}
E & \to & TE' \\
E' & \to & +TE' \mid \epsilon \\
T & \to & FT' \\
T' & \to & *FT' \mid \epsilon \\
F & \to & (E) \mid \textbf{id}
\end{array}
$$

$\texttt{first}_1(E) = \{(, \textbf{id}\}$ $\quad\texttt{first}_1(E') = \{+, \epsilon\}$
$\texttt{first}_1(T) = \{(, \textbf{id}\}$ $\quad\texttt{first}_1(T') = \{*, \epsilon\}$
$\texttt{first}_1(F) = \{(, \textbf{id}\}$

$\texttt{follow}_1(E) = \{), \#\}$ $\quad\texttt{follow}_1(E') = \{), \#\}$
$\texttt{follow}_1(T) = \{+,), \#\}$ $\quad\texttt{follow}_1(T') = \{+,), \#\}$
$\texttt{follow}_1(F) = \{+, *,), \#\}$

	id	+	*	(	)	#
E	TE'	ERROR	ERROR	TE'	ERROR	ERROR
E'	ERROR	$+TE'$	ERROR	ERROR	ϵ	ϵ
T	FT'	ERROR	ERROR	FT'	ERROR	ERROR
T'	ERROR	ϵ	$*FT'$	ERROR	ϵ	ϵ
F	**id**	ERROR	ERROR	(E)	ERROR	ERROR

6. Extracción de información

6.1. Extracción

6.2. Enumeración de resultados: Autómatas con anotaciones