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

5 de diciembre de 2022



Índice

1. Lenguajes regulares **3**
- 1.1. Palabras y autómatas . . . 3
  - 1.1.1. Palabras . . . 3
  - 1.1.2. Autómatas . . . 4
- 1.2. Construcciones de autómatas . . . 6
  - 1.2.1. Autómatas con función parcial de transición . . . 6
  - 1.2.2. Operaciones de conjuntos . . . 7
- 1.3. No-determinismo . . . 9
  - 1.3.1. Definición de un NFA . . . 9
  - 1.3.2. Comparación con DFA . . . 11
- 1.4. Expresiones regulares . . . 13
- 1.5. Autómatas con transiciones sin lectura . . . 15
  - 1.5.1. e-NFA . . . 15
  - 1.5.2. NFA versus e-NFA . . . 17
- 1.6. Teorema de Kleene . . . 19
  - 1.6.1. Desde Expresiones a Autómatas . . . 19
  - 1.6.2. Desde Autómatas a Expresiones . . . 20

2. Propiedades de lenguajes regulares **23**
- 2.1. Lema de bombeo . . . 23
- 2.2. Minimización de autómatas . . . 26
  - 2.2.1. Colapsar estados . . . 27
  - 2.2.2. Algoritmo de minimización . . . 28
- 2.3. Teorema de Myhill-Nerode . . . 30
  - 2.3.1. Relaciones de Myhill-Nerode . . . 30
  - 2.3.2. Camino al teorema . . . 31
- 2.4. Autómatas en dos direcciones . . . 33
  - 2.4.1. Definición de un 2DFA . . . 33
  - 2.4.2. 2DFA vs DFA . . . 34

3. Algoritmos para lenguajes regulares **37**
- 3.1. Evaluación de expresiones regulares . . . 37
- 3.2. Transductores . . . 40
- 3.3. Análisis léxico . . . 43
- 3.4. Algoritmo de Knuth-Morris-Prat . . . 43
  - 3.4.1. Autómata de un patrón . . . 43
  - 3.4.2. Autómata finito con k-lookahead . . . 45
  - 3.4.3. Algoritmo KMP . . . 46

4. Lenguajes libres de contexto **48**
- 4.1. Gramáticas libres de contexto . . . 48
  - 4.1.1. Gramáticas . . . 48
  - 4.1.2. Árboles y derivaciones . . . 50
  - 4.1.3. Lenguajes regulares vs libres de contexto . . . 53
- 4.2. Simplificación de gramáticas . . . 53
  - 4.2.1. Eliminación de variables inútiles . . . 53
  - 4.2.2. Eliminación de producciones inútiles . . . 55
- 4.3. Forma normal de Chomsky . . . 56
- 4.4. Lema de bombeo para lenguajes libres de contexto . . . 58
- 4.5. Algoritmo CKY . . . 61

5. Algoritmos para lenguajes libres de contexto **64**
5.1. Autómatas apiladores . . . 64
5.1.1. Versión normal . . . 64
5.1.2. Versión alternativa . . . 66
5.2. Autómatas apiladores vs gramáticas libres de contexto . . . 68
5.2.1. Desde CFG a PDA . . . 69
5.2.2. Desde PDA a CFG . . . 70
5.3. Parsing: cómputo de First y Follow . . . 72
5.3.1. Prefijos . . . 74
5.3.2. First y Follow . . . 75
5.3.3. Calcular First . . . 76
5.3.4. Calcular Follow . . . 77
5.4. Gramáticas LL . . . 78
5.4.1. Definición Gramáticas LL . . . 79
5.4.2. Caracterización LL . . . 81
5.5. Parsing con gramáticas LL(k) . . . 83
5.5.1. Algunas consideraciones . . . 83
5.5.2. Parsing de LL(k) . . . 86

6. Extracción de información **90**
6.1. Extracción . . . 90
6.1.1. Spans . . . 90
6.1.2. Regex . . . 91
6.1.3. Vset autómata . . . 93
6.1.4. Desde regex a VA . . . 96
6.2. Enumeración de resultados: Autómatas con anotaciones . . . 96
6.2.1. Representación de mappings . . . 96
6.2.2. Autómatas con anotaciones . . . 97
6.2.3. Desde un vset a AnnA . . . 99
6.2.4. Determinismo . . . 101

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.

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

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

$$|\epsilon| \stackrel{\text{def}}{\equiv} 0$$

- 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 $v = b_1 \dots v_m$:

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

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:

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

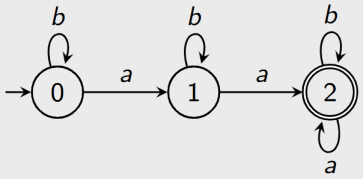
- Q es un conjunto finito de **estados**.
- Σ es el alfabeto del **input**.
- $\delta : Q \times \Sigma \to Q$ es la función de **transición**.
- $q_0 \in Q$ es el **estado inicial**.
- $F \subseteq Q$ es el conjunto de **estados finales** (o aceptación).

Ejemplo 1.5

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

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

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



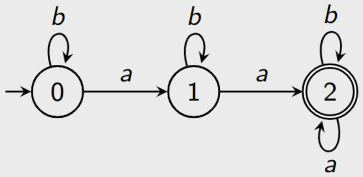
Ejecución. Sea $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un DFA y $w = a_1 a_2 \dots a_n \in \Sigma^*$ un input. Una **ejecución** (o *run*) ρ de $\mathcal{A}$ sobre w es una secuencia:

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

donde $p_0 = q_0$ y para todo $i \in \{0, 1, \dots, n-1\}$ 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:

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

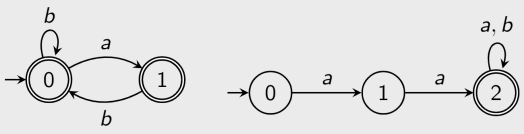
donde $p_0 = q_0$ y para todo $i \in \{0, \dots, n-1\}$ **está definido** $\gamma(p_i, a_{i+1}) = p_{i+1}$.

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 $\equiv$ 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:

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

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:

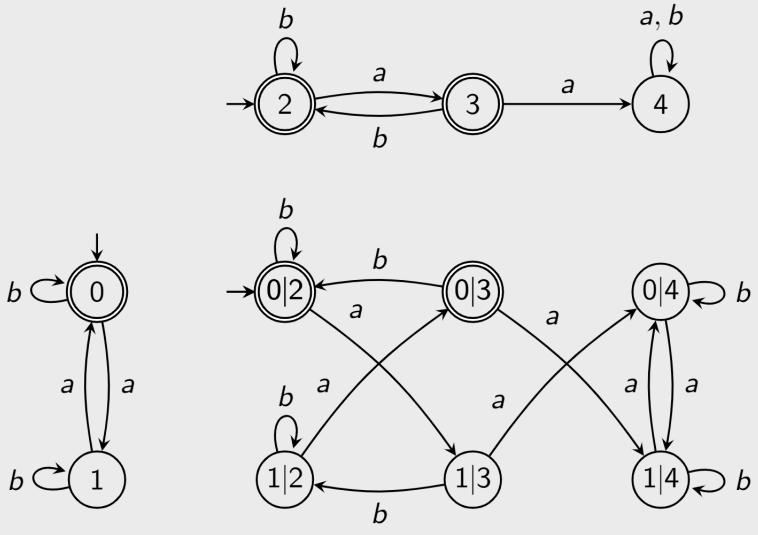
$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$$
$$\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$$

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:

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n \quad \rho' : p'_0 \xrightarrow{a_1} p'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p'_n$$

- $p_0 = q_0$ 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: $\rho^\times : (p_0, p'_0) \xrightarrow{a_1} (p_1, p'_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (p_n, p'_n)$

- $(p_0, p'_0) = (q_0, q'_0)$.
- $(p_i, p'_i) = \left(\delta\left(p_{i-1}, a_i\right), \delta'\left(p'_{i-1}, a_i\right)\right) = \delta^\times\left(\left(p_{i-1}, p'_{i-1}\right), a_i\right) \forall i \in \{1, \dots, n\}$.
- $(p_n, p'_n) \in F \times F'$.

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:

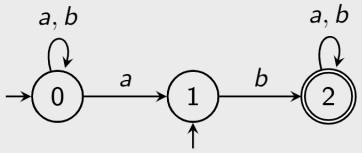
$$\boxed{\mathcal{A} = (Q, \Sigma, \Delta, I, F)}$$

- 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:

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

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:

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

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.

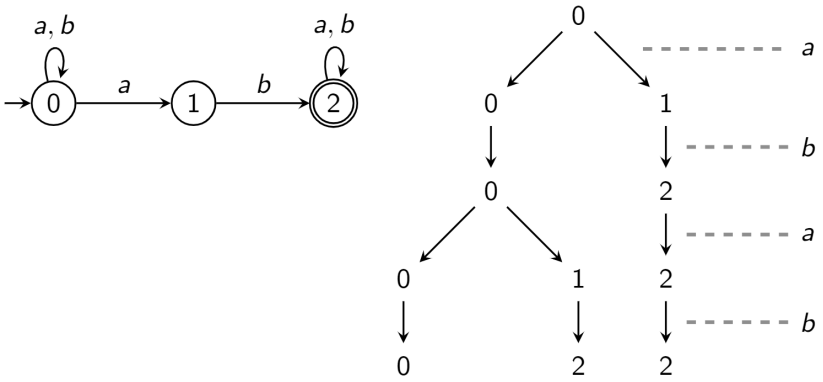


Figura 1: Interpretación del no-determinismo

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

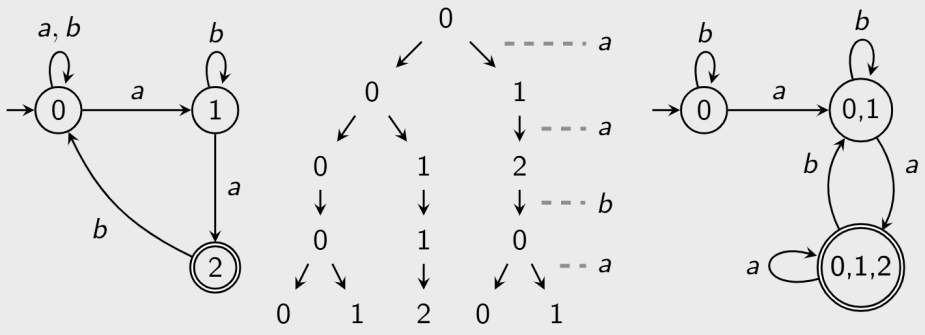
Para todo autómata finito no-determinista $\mathcal{A}$, existe un autómata determinista $\mathcal{A}'$ tal que

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

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 $\mathcal{A}$):

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

- $2^Q = \{S \mid S \subseteq Q\}$ 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. $\mathcal{L}(\epsilon) = \{\epsilon\}$.
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:

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

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

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

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**.

$$(R_1 + R_2) + R_3 \equiv R_1 + (R_2 + R_3)$$
$$(R_1 \cdot R_2) \cdot R_3 \equiv R_1 \cdot (R_2 \cdot R_3)$$

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:

$$R^+ \equiv R \cdot R^*$$
$$R^k \equiv R \cdot \overset{k}{\cdots} \cdot R$$
$$R^? \equiv R + \epsilon$$
$$\Sigma \equiv a_1 + \ldots + a_n$$

para $R \in \text{ExpReg}$ 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:

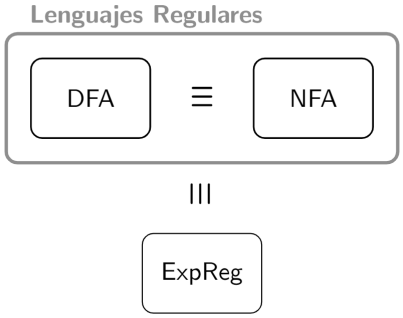


Figura 2: Mapa actual de nuestros modelos de computación

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:

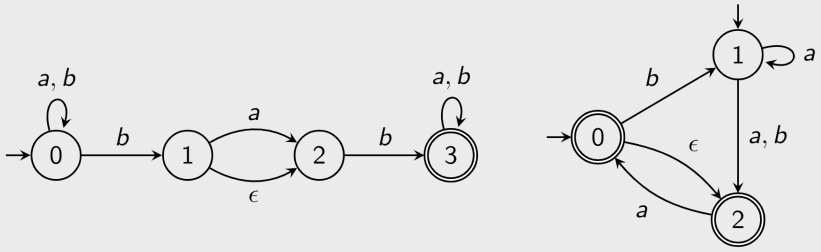
$$\boxed{\mathcal{A} = (Q, \Sigma, \Delta, I, F)}$$

- 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 $\mathcal{A}$:

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

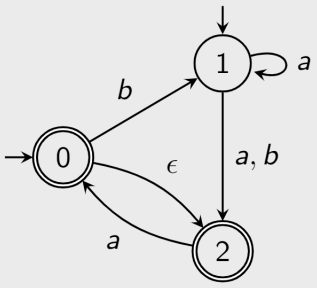
  si, y sólo si, existe $(p, c, q) \in \Delta$ con $c \in \Sigma \cup \{\epsilon\}$ tal que $u = c \cdot v$.

- 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



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

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

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

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:

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

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

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

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:

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

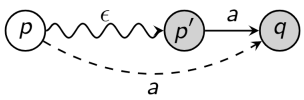
En otras palabras, NFA $\equiv$ ϵ-NFA.

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:

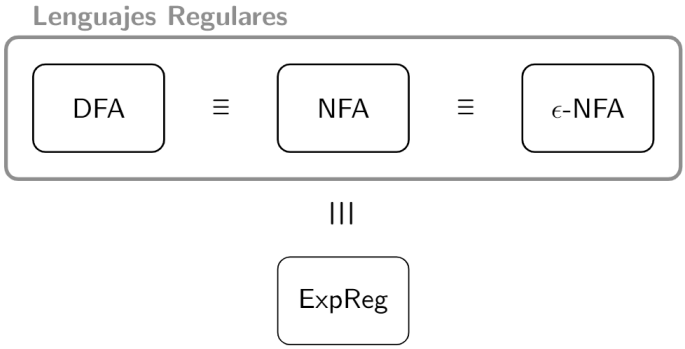


Figura 3: Mapa actual de nuestros modelos de computación

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.

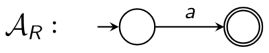
Construcción inductiva. Para cada $R \in \text{ExpReg}$, construimos un ϵ-NFA $\mathcal{A}_R$:

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

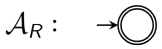
tal que $\mathcal{L}(R) = \mathcal{L}(\mathcal{A}_R)$.

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

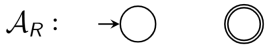
1. si $R = a$, para alguna letra $a \in \Sigma$:



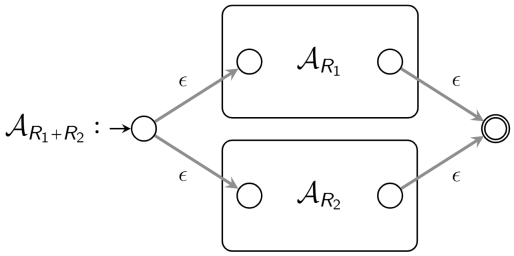
2. si $R = \epsilon$:



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 $R = (R_1 + R_2)$. Sea $\mathcal{A}_{R_1}$ y $\mathcal{A}_{R_2}$ los ϵ-NFA para R_1 y R_2, respectivamente, tal que:

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

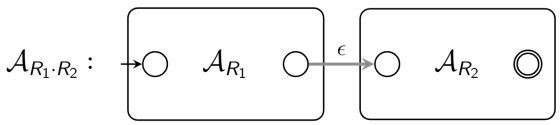
Definimos el ϵ-NFA $\mathcal{A}_{R_1+R_2} = (Q, \Sigma, \Delta, \{q_0\}, \{q_f\})$ tal que:

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

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

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 $R = (R_1 \cdot R_2)$. Sea $\mathcal{A}_{R_1}$ y $\mathcal{A}_{R_2}$ los ϵ-NFA para R_1 y R_2, respectivamente, tal que:

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

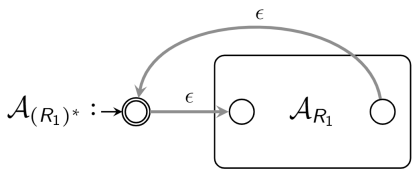
Definimos el ϵ-NFA $\mathcal{A}_{R_1 \cdot R_2} = (Q, \Sigma, \Delta, \{q_0^1\}, \{q_f^2\})$ tal que:

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

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

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 $R = (R_1^*)$. 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:

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

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

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, $ExpReg \subseteq \epsilon$-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 $w = a_1 \dots a_n \in \alpha_{p,q}^X$ si, y sólo si, existe una **ejecución**:

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

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, $\alpha^X_{p,q}$ 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 $\alpha^X_{p,q}$? Establecemos el siguiente lema:

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

Estrategia. Conocida como el algoritmo de McNaughton-Yamada:

1. Para cada $\alpha^X_{p,q}$, definir **inductivamente** una expresión regular $R^X_{p,q}$:
$$\mathcal{L}(R^X_{p,q}) = \alpha^X_{p,q}$$
2. Para $F = \{p_1, \ldots, p_k\}$ definir la **expresión regular**:
$$R_{\mathcal{A}} = R^Q_{q_0,p_1} + R^Q_{q_0,p_2} + \ldots + R^Q_{q_0,p_k}$$
3. Demostrar que:
$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(R_{\mathcal{A}})$$

Definición inductiva de $R^X_{p,q}$. 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:
  $$R^{\varnothing}_{p,q} \stackrel{\text{def}}{\equiv} \begin{cases} a_1 + \cdots + a_k & \text{si } k \geq 1 \\ \varnothing & \text{si } k = 0 \end{cases}$$
  - Si $p = q$, entonces:
  $$R^{\varnothing}_{p,q} \stackrel{\text{def}}{\equiv} \begin{cases} a_1 + \cdots + a_k + \epsilon & \text{si } k \geq 1 \\ \epsilon & \text{si } k = 0 \end{cases}$$

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

  Por **inducción**, suponemos que para todo $r, s \in Q$ y para todo $Y \subset X$, $R^Y_{r,s}$ es una expresión regular tal que:
  $$\mathcal{L}(R^Y_{r,s}) = \alpha^Y_{r,s}$$
  Demostramos la construcción para $R^X_{p,q}$ con $p, q \in Q$. Sea $r \in X$ cualquiera:
  $$R^X_{p,q} \stackrel{\text{def}}{\equiv} R^{X-\{r\}}_{p,q} + R^{X-\{r\}}_{p,r} \cdot \left(R^{X-\{r\}}_{r,r}\right)^* \cdot R^{X-\{r\}}_{r,q}$$

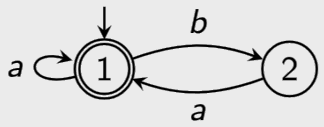
Proposición. Para todo $X \subseteq Q$ y $p, q \in Q$:

$$\mathcal{L}\left(R^X_{p,q}\right) = \alpha^X_{p,q}$$

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	$a^* + a^*b\left(\epsilon + a^+b\right)^* a^+$	$a^*b + a^*b\left(\epsilon + a^+b\right)^* \left(\epsilon + a^+b\right)$
2	$a^+ + \left(\epsilon + a^+b\right)\left(\epsilon + a^+b\right)^* a^+$	$\left(\epsilon + a^+b\right) + \left(\epsilon + a^+b\right)\left(\epsilon + a^+b\right)^* \left(\epsilon + a^+b\right)$

$\equiv$

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

2. Propiedades de lenguajes regulares

2.1. Lema de bombeo

Supongamos que deseamos aceptar el siguiente lenguaje:

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

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:

> (¬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 $|y| \geq N$

El escoge $u \cdot v \cdot w = y$ con $v \neq \epsilon$

Uno escoge $i \geq 0$

Uno gana si $xuv^i wz \notin L$ — **El gana** si $xuv^i wz \in L$

LB versión juego. *"Dado un lenguaje $L \subseteq \Sigma^*$, si **UNO** tiene una estrategia ganadora en el juego (¬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?

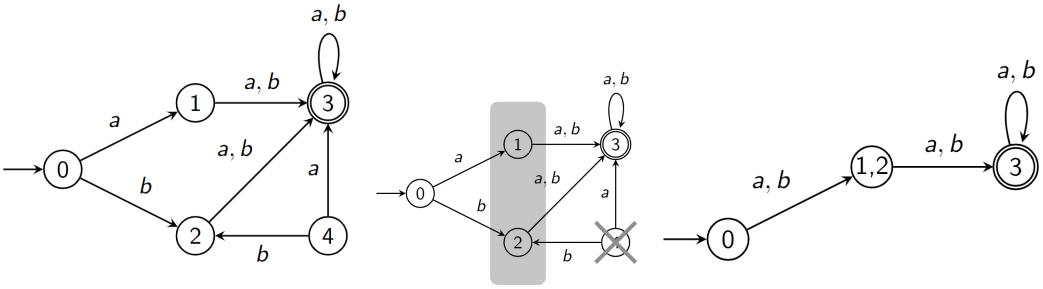
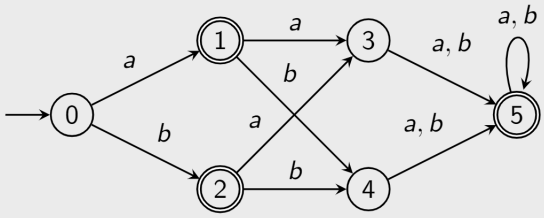


Figura 4: Idea de minimización

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?

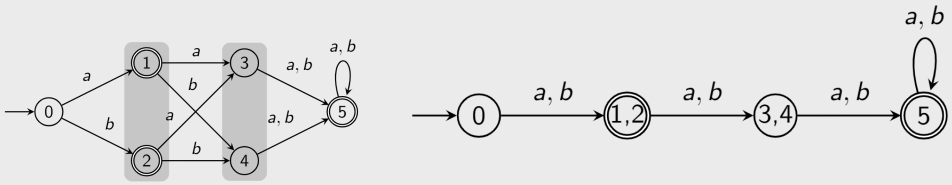
Ejemplo 2.5

Considere el siguiente autómata:



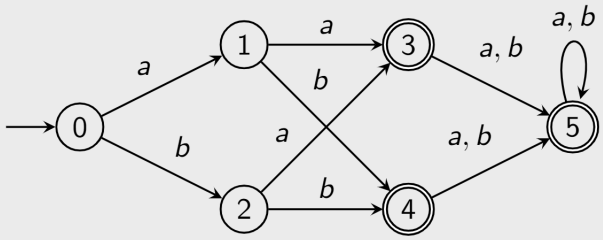
Podemos:

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



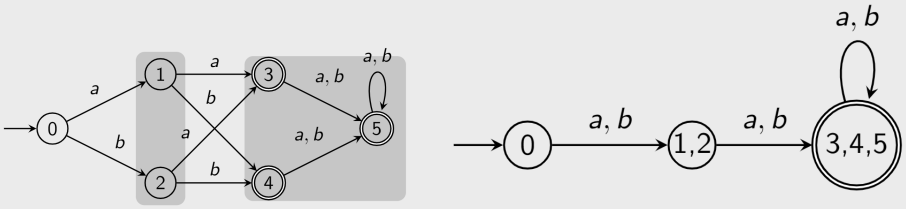
Ejemplo 2.6

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:

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

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

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

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:

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

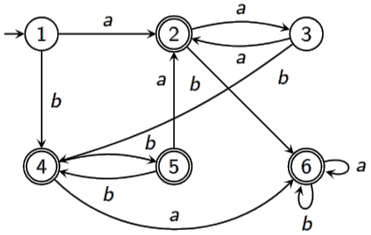
Demostración. Queda como ejercicio propuesto al lector.

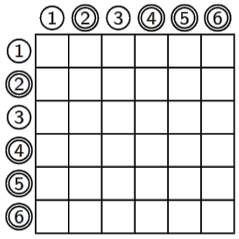
2.2.2. Algoritmo de minimización

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

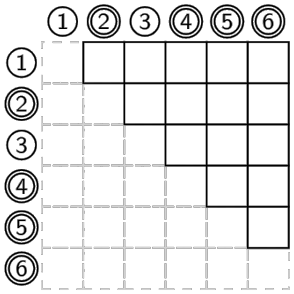
1. Construya una tabla con los pares $\{p, q\}$ inicialmente sin marcar.
2. Marque $\{p, q\}$ si $p \in F$ y $q \notin F$ o viceversa.
3. Repita este paso hasta que no hayan más cambios:
   - Si $\{p, q\}$ no están marcados y $\{\delta(p, a), \delta(q, a)\}$ estan marcados para algún $a \in \Sigma$,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

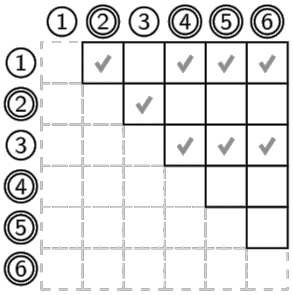




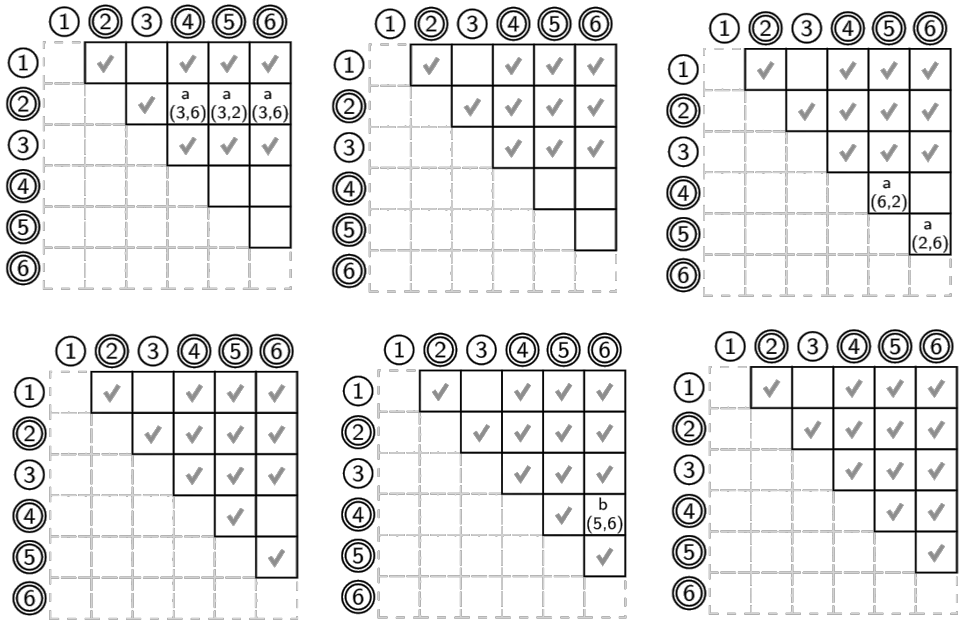
1. Construya una tabla con los pares $\{p, q\}$ inicialmente sin marcar.



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, $p \not\approx_{\mathcal{A}} q$ 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 $\mathcal{A}_{\equiv}$.

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

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:

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

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**:

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

■

2.4. Autómatas en dos direcciones

2.4.1. Definición de un 2DFA

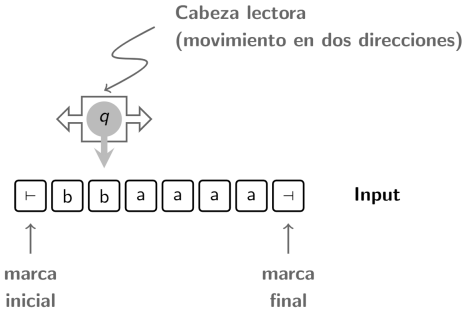


Figura 5: Representación de un 2DFA

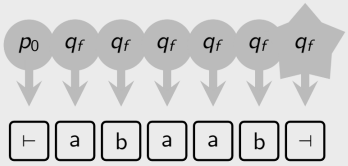
Definición. Un autómata finito determinista en 2 direcciones (2DFA) es una estructura:

$$\mathcal{A} = (Q, \Sigma, \vdash, \dashv, \delta, q_0, q_f)$$

- Q es un conjunto finito de estados.
- Σ es el alfabeto del input.
- $\vdash$ y $\dashv$ son las marcas (símbolos) iniciales y finales.
- $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{\leftarrow, \rightarrow\}$ es la **función parcial de transición**.
- q_0 es el estado inicial.
- q_f es el estado final.

Ejemplo 2.8

$$L = \left\{ w \in \Sigma^* \mid \left(\#a(w) = 0 \mod 3\right) \text{ y } \left(\#b(w) = 0 \mod 2\right) \right\}$$



	$\vdash$	a	b	$\dashv$
q_0	$(q_0, \rightarrow)$	$(q_1, \rightarrow)$	$(q_0, \rightarrow)$	$(p_0, \leftarrow)$
q_1	-	$(q_2, \rightarrow)$	$(q_1, \rightarrow)$	-
q_2	-	$(q_0, \rightarrow)$	$(q_2, \rightarrow)$	-
p_0	$(q_f, \rightarrow)$	$(p_0, \leftarrow)$	$(p_1, \leftarrow)$	-
p_1	-	$(p_1, \leftarrow)$	$(p_0, \leftarrow)$	-
q_f	-	$(q_f, \rightarrow)$	$(q_f, \rightarrow)$	-

Configuración. Sea $\mathcal{A}$ un 2DFA y $w = a_1 \dots a_n \in \Sigma^*$ un input. Definimos $a_0 = \vdash$ y $a_{n+1} = \dashv$ tal que el input se define como:

$$a_0 a_1 \dots a_n a_{n+1} = \vdash \cdot\, w \,\cdot\, \dashv$$

Una **configuración** de $\mathcal{A}$ sobre w viene dado por un par:

$$(q, i) \in Q \times \{0, \dots, n+1\}$$

- q es el **estado actual** del autómata.
- i es la **posición actual** de la cabeza lectora.

Se define la relación de **siguiente configuración** $\overset{\mathcal{A}}{\longmapsto}$ de $\mathcal{A}$ sobre w como:

$$(p, i) \overset{\mathcal{A}}{\longmapsto} (q, j)$$

tal que:

- Si $\delta(p, a_i) = (q, \rightarrow)$, entonces $(p, i) \overset{\mathcal{A}}{\longmapsto} (q, i+1)$.
- Si $\delta\,(p, a_i) = (q, \leftarrow)$, entonces $(p, i) \overset{\mathcal{A}}{\longmapsto} (q, i-1)$.

Ejecución. Una ejecución (o *run*) ρ de $\mathcal{A}$ sobre w es una secuencia de configuraciones:

$$\rho : (p_0, i_0) \to (p_1, i_1) \to \dots \to (p_m, i_m)$$

donde $p_0 = q_0$ y $i_0 = 0$. Además, $(p_j, i_j) \overset{\mathcal{A}}{\longmapsto} (p_{j+1}, i_{j+1}) \quad \forall j \in [0, m-1]$.

Una ejecución ρ de $\mathcal{A}$ sobre w es de **aceptación** si:

$$p_m = q_f \quad y \quad i_m = n+1$$

Aceptación. Decimos que $\mathcal{A}$ **acepta** w si hay una ejecución de $\mathcal{A}$ sobre w que es de **aceptación**. Así, el **lenguaje aceptado** por $\mathcal{A}$ se define como:

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

Ojo: Notemos que un 2DFA puede pasar por error o NO parar nunca.

2.4.2. 2DFA vs DFA

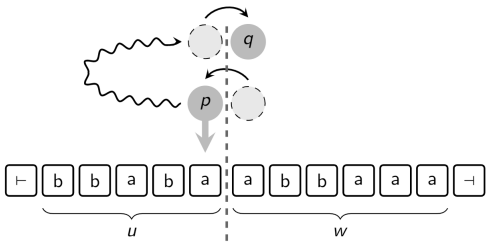
Para todo lenguaje regular L existe un 2DFA $\mathcal{A}$:

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

En otras palabras, DFA $\subseteq$ 2DFA.

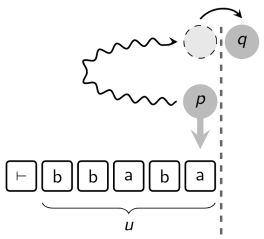
¿Son los 2DFA más poderosos que los DFA? En esta sección, demostraremos que no, es decir, pueden representar al mismo conjunto de lenguajes.

¿Cuánta información puede almacenar un 2DFA? Veamos la siguiente figura:



"Cada vez que $\mathcal{A}$ cruce de w a u en el estado p, $\mathcal{A}$ cruzará de regreso en el estado q".

Este comportamiento solo depende de u, y no de w:

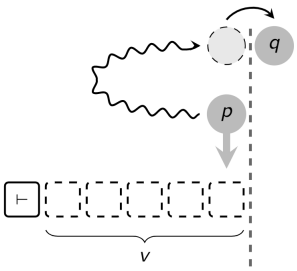


Para cada $u \in \Sigma^*$, definimos la función $T_u : Q \cup \{\bullet\} \to Q \cup \{\perp\}$ tal que:

- $T_u(p) = q$ si, y sólo si, desde $(p, |u|)$ cruza en la configuración $(q, |u| + 1)$.
- $T_u(p) = \perp$ si, y sólo si, desde $(p, |u|)$ nunca cruza de u.
- $T_u(\bullet) = q$ si, y sólo si, desde $(q_0, 0)$ cruza por primera vez con $(q, |u| + 1)$.
- $T_u(\bullet) = q$ si, y sólo si, desde $(q_0, 0)$ cruza por primera vez con $(q, |u| + 1)$.
- $T_u(\bullet) = \perp$ si, y sólo si, desde $(q_0, 0)$ nunca cruza de u.

Suponga que ahora tenemos una palabra v tal que:

$$T_v = T_u$$



Entonces, v es **indistingible** de u según $\mathcal{A}$. En otras palabras:

$$u \cdot w \in \mathcal{L}(\mathcal{A}) \Leftrightarrow v \cdot w \in \mathcal{L}(\mathcal{A}) \quad \text{para todo } w \in \Sigma^*$$

Lo anterior tiene conexión con las relaciones de Myhill-Nerode, que vimos en una sección anterior.

Luego, respecto a la función T_u podemos decir que:

1. Definimos la relación $\equiv_T$ entre palabras en Σ^* tal que:

   $$u \equiv_T v \text{ si, y solo si, } \quad T_u = T_v$$

   es una **relación de equivalencia**.

2. $\equiv_T$ es una **congruencia por la derecha**:

   $$u \equiv_T v \Rightarrow \forall w \cdot u \cdot w \equiv_T v \cdot w$$

3. $\equiv_T$ **refina** a $\mathcal{L}(\mathcal{A})$:

   $$u \equiv Tv \Rightarrow (u \in L \Leftrightarrow v \in L)$$

4. La relación $\equiv_T$ tiene una **cantidad finita** de clases de equivalencia:

   $$T : Q \cup \{\bullet\} \to Q \cup \{\perp\}$$

   Así, $\equiv_{\mathcal{L}(\mathcal{A})}$ también tiene una cantidad **finita** de **clases de equivalencia**.

Teorema 11

Para todo 2DFA $\mathcal{A}$ existe un DFA $\mathcal{A}'$ tal que:

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

En otras palabras, 2DFA $\equiv$ DFA.

El DFA del teorema anterior se construye usando el Teorema de Myhill-Nerode a partir de las funciones T_u.

La demostración del teorema y la construcción queda como ejercicio propuesto para el lector.

3. Algoritmos para lenguajes regulares

3.1. Evaluación de expresiones regulares

Expresiones regulares en la práctica. El lenguaje para definir expresiones regulares en la práctica se conoce como **RegEx** (o RexExp). Las sintaxis más usadas para definir **RegEx** son:

1. POSIX (Portable Operating System Interface for uniX).
2. Perl (PCRE = Perl Compatible Regular Expressions).

	RegEx	Teoría
Carácter	`a`	a
Escape	\+	$-$
Cualquiera	·	Σ
Clase	`[abc]`	$(a+b+c)$
Clase consecutivo	`[a-zA-Z]`	$(a+\cdots+z+A+\cdots+Z)$
Clase exclusivo	`[^0-9]`	$\left(+ +_{a\in\Sigma-\{0,\ldots,9\}} \text{a}\right)$
Alternación	`cat` | `dog`	cat + dog
0 o 1	`R?`	$R^?$
1 o más	`R+`	R^+
0 o más	`R*`	R^*
entre n y m	`R{n,m}`	$R^n(R+\epsilon)^{m-n}$
Backreference	(R)\1	?
...	...	...

¿Cómo evaluamos una expresión regular? Tenemos el siguiente problema:

`PROBLEMA:`	Evaluación de expresiones regulares
`INPUT:`	una expresión regular R
	un documento w
`OUTPUT:`	`TRUE` si, y sólo si, $w \in \mathcal{L}(R)$

Donde el tamaño del input está dado por:

- $|R|$:= número de letras y operadores.
- $|w|$:= largo del documento.

Buscamos un algoritmo polinomial en $|R|$ y $|w|$. Algo a notar es que $|R| << |w|$:

- $|R|$ puede ser del orden **decenas** operadores ($\sim$ 1 KB).
- $|w|$ puede ser del orden de **miles a millones** de símbolos ($\sim$ 100MB).

Análisis de tiempo diferenciado. Tenemos dos tipos:

- **Combined complexity:** expresión y documento son parte del input.
- **Data complexity:** solo documento es parte del input (tamaño de expresión es considerada una constante).

Buscamos algoritmos que sean **lineales en data complexity** y ojalá **polinomiales en combined complexity**.

Volviendo a nuestro problema:

`PROBLEMA:`	Evaluación de expresiones regulares
`INPUT:`	una expresión regular R
	un documento w
`OUTPUT:`	`TRUE` si, y sólo si, $w \in \mathcal{L}(R)$

Los pasos que podemos seguir para resolverlo son:

1. Convertimos R a un ϵ-NFA $\mathcal{A}_R$.
2. Verificamos si $w \in \mathcal{L}(\mathcal{A}_R)$.

Ahora, el tamaño del input es:

- $|w|$:= largo del documento.
- $|\mathcal{A}| := |Q| + |\Delta|$.

Hay varias soluciones para resolver nuestro problema, que iremos enumerado a continuación.

Solución 1: Backtracking. Esta solución es la que usa la mayoría de los motores de RegEx en la práctica. Se puede hacer una prueba en `https://regexr.com/`.

Solución 2: DFA. Para un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ y una palabra $w = a_1 \dots a_n$.

Function `eval-DFA(`$\mathcal{A}$, w`)`**:**
  $q := q_0$
  for $i = 1$ **to** n **do**
    $q := \delta(q, a_i)$
  return check $(q \in F)$

Solución 3: NFA determinización. Para un NFA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ y una palabra $w = a_1 \dots a_n$.

Function `eval-NFA(`$\mathcal{A}$, w`)`**:**
  $\mathcal{A}^{\text{det}} :=$ `NFAtoDFA(`$\mathcal{A}$`)`
  return `eval-DFA(`$\mathcal{A}^{det}, w$`)`

¿Es necesario construir la determinización completa? La respuesta es que no. Recordemos que es la determinización. 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.

- $q_0^{\det} = I$.
- $\delta^{\det} : 2^Q \times \Sigma \to 2^Q$ tal que:

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

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

Solución 4: NFA *on-the-fly*. Fijémonos en la función de transición $\delta^{\det} : 2^Q \times \Sigma \to 2^Q$ tal que:

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

La estrategia *on-the-fly* corresponde a:

1. Mantenemos un conjunto S de estados actuales.
2. Por cada nueva letra a, calculamos el conjunto $\delta^{\det}(S, a)$.

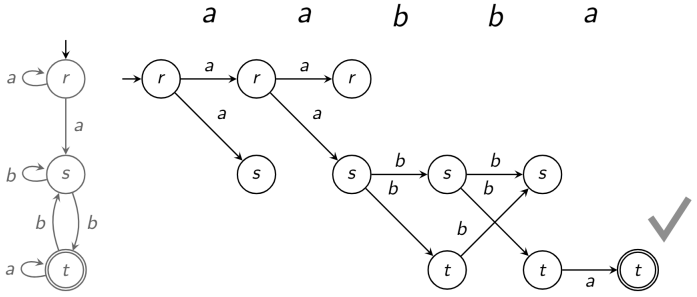


Figura 6: Ejemplo de NFa *on-the-fly*

Definimos el algoritmo para un NFA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ y una palabra $w = a_1 \dots a_n$ como:

Function `eval-NFAonthefly`($\mathcal{A}$, w)**:**
  $S := I$
  for $i = 1$ **to** n **do**
    $S_{\text{old}} := S$
    $S := \varnothing$
    foreach $p \in S_{old}$ **do**
      $S := S \cup \{q \mid (p, a_i, q) \in \Delta\}$
  return check $(S \cap F \neq \varnothing)$

Resumen y complejidades. Para un autómata $\mathcal{A}$ y una palabra $w = a_1 \dots a_n$ tenemos que:

	Tiempo
Backtracking	$\mathcal{O}\left(\lvert\mathcal{A}\rvert^{\lvert w\rvert}\right)$
DFA	$\mathcal{O}(\lvert\mathcal{A}\rvert + \lvert w\rvert)$
NFA	$\mathcal{O}\left(2^{\lvert Q\rvert} + \lvert w\rvert\right)$
ϵ-NFA *on-the-fly*	$\mathcal{O}(\lvert\mathcal{A}\rvert \cdot \lvert w\rvert)$

3.2. Transductores

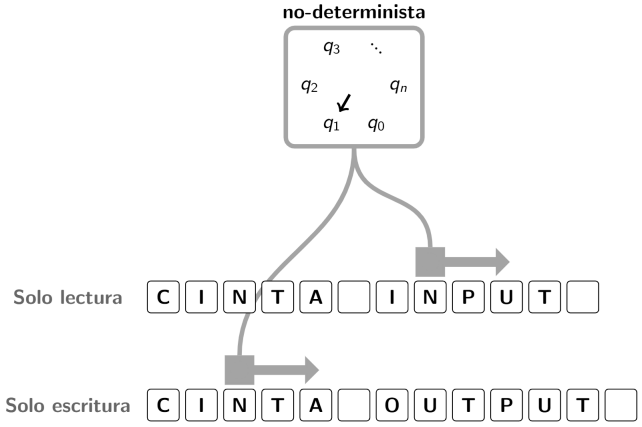


Figura 7: Representación de un transductor

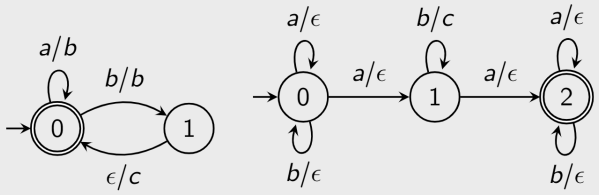
Definición. Un transductor (en inglés, *transducer*) es una tupla:

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

- Q es un conjunto finito de estados.
- Σ es el alfabeto del input.
- Ω es el alfabeto del **output**.
- $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times Q$ es la **relación de transición**.
- $I \subseteq Q$ es un conjunto de estados iniciales.
- $F \subseteq Q$ es el conjunto de estados finales.

Ejemplo 3.1

Algunos transductores:



Configuración. Sea $\mathcal{T}$ un transductor. Definimos:

- Un par $(q, u, v) \in Q \times \Sigma^* \times \Omega^*$ es una **configuración** de $\mathcal{T}$.
- Una configuración (q, u, ϵ) es **inicial** si $q \in F$.
- Una configuración (q, ϵ, v) es **final** si $q \in F$.

"Intuitivamente, una configuración (q, au, vb) representa que $\mathcal{T}$ se encuentra en el estado q procesando la palabra au y leyendo a, y hasta ahora grabó la palabra vb y el último símbolo impreso es b".

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

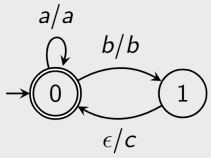
$$\boxed{(p, u_1, v_1) \vdash_{\mathcal{T}} (q, u_2, v_2)}$$

si, y sólo si, existe $(p, a, b, q) \in \Delta$ tal que $u_1 = a \cdot u_2$ y $v_2 = v_1 \cdot b$.

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

$$\boxed{\begin{array}{rl} \text{para toda configuración } (q, u, v): & (q, u, v) \vdash^*_{\mathcal{T}} (q, u, v) \\ \text{si } (q_1, u_1, v_1) \vdash^*_{\mathcal{T}} (q_2, u_2, v_2) \text{ y } (q_2, u_2, v_2) \vdash_{\mathcal{T}} (q_3, u_3, v_3): & (q_1, u_1, v_1) \vdash^*_{\mathcal{T}} (q_3, u_3, v_3) \end{array}}$$

Ejemplo 3.2



$(0, aba, \epsilon) \vdash_{\mathcal{T}} (0, ba, a)$

$(0, ba, a) \vdash_{\mathcal{T}} (1, a, ab) \vdash_{\mathcal{T}} (0, a, abc) \vdash_{\mathcal{T}} (0, \epsilon, abca)$

$(0, aba, \epsilon) \vdash^*_{\mathcal{T}} (0, ba, a)$ y $(0, aba, \epsilon) \vdash^*_{\mathcal{T}} (0, \epsilon, abca)$

Función definida por un transductor. Dado un transductor $\mathcal{T}$, decimos que:

- $\mathcal{T}$ **entrega** v con **input** u si existe una configuración inicial (q_0, u, ϵ) y una configuración final (q_f, ϵ, v) tal que:

$$\boxed{(q_0, u, \epsilon) \vdash^*_{\mathcal{T}} (q_f, \epsilon, v)}$$

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

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

- Se dice que $f : \Sigma^* \to 2^{\Omega^*}$ es una **función racional** si existe un transductor $\mathcal{T}$ tal que $f = [\![\mathcal{T}]\!]$.
- Un transductor define una función de palabras a conjuntos de palabras.

Dos interpretaciones para un transductor. Podemos ver a $[\![\mathcal{T}]\!]$ de dos formas:

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

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

2. $\mathcal{T}$ define la **relación** $[\mathcal{T}] \subseteq \Sigma^* \times \Omega^*$:

$$(u, v) \in [\![\mathcal{T}]\!] \quad \text{si, y solo si,} \quad \mathcal{T} \text{ entrega } v \text{ con input } u$$

Desde ahora, hablaremos de función o relación **indistintamente** y hablaremos de las **relaciones racionales** (definidas por un transductor).

Lenguaje de input y output. Para una relación $R \subseteq \Sigma^* \times \Omega^*$ se define:

- $\pi_1(R) = \{u \in \Sigma^* \mid \exists v \in \Omega^*.\ (u, v) \in R\}$.
- $\pi_2(R) = \{v \in \Omega^* \mid \exists u \in \Sigma^*.\ (u, v) \in R\}$.

Teorema 12

Si $\mathcal{T}$ es un transductor, entonces $\pi_1([\![\mathcal{T}]\!])$ y $\pi_2([\![\mathcal{T}]\!])$ son lenguajes regulares sobre Σ y Ω, respectivamente.

Idea demostración teorema 12. Para $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$, defina $\mathcal{A}_1 = (Q, \Sigma, \Delta_1, I, F)$ tal que

$$(p, a, q) \in \Delta_1 \quad \text{si, y solo si,} \quad \exists b \in \Omega \cup \{\epsilon\}.\ (p, a, b, q) \in \Delta$$

y demuestre que $\mathcal{L}(\mathcal{A}_1) = \pi_1([\![\mathcal{T}]\!])$.

Teorema 13

*Sea $\mathcal{T}_1$ y $\mathcal{T}_2$ dos transductores con Σ y Ω alfabetos de input y output. Las siguientes son **relaciones racionales**:*

1. $[\![\mathcal{T}_1]\!] \cup [\![\mathcal{T}_2]\!] = \{(u, v) \in \Sigma^* \times \Omega^* \mid (u, v) \in [\![\mathcal{T}_1]\!] \vee (u, v) \in [\![\mathcal{T}_2]\!]\}$
2. $[\![\mathcal{T}_1]\!] \cdot [\![\mathcal{T}_2]\!] = \{(u_1u_2, v_1v_2) \in \Sigma^* \times \Omega^* \mid (u_1, v_1) \in [\![\mathcal{T}_1]\!] \wedge (u_2, v_2) \in [\![\mathcal{T}_2]\!]\}$
3. $[\![\mathcal{T}_1]\!]^* = \bigcup_{k=0}^{\infty} [\![\mathcal{T}_1]\!]^k$

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

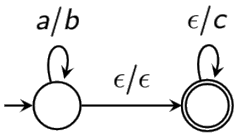
Teorema 14

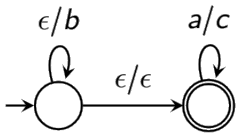
Existen transductores $\mathcal{T}_1$ y $\mathcal{T}_2$ sobre Σ y Ω tal que:

$$[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!] = \{(u, v) \in \Sigma^* \times \Omega^* \mid (u, v) \in [\![\mathcal{T}_1]\!] \wedge (u, v) \in [\![\mathcal{T}_2]\!]\}$$

NO es una relación racional.

Demostración teorema 14. Considere los siguientes transductores:





$[\![\mathcal{T}_1]\!] = \{(a^n, b^n c^m) \mid n \geq 0, m \geq 0\}$ $\quad$ $[\![\mathcal{T}_2]\!] = \{(a^n, b^m c^n) \mid n \geq 0, m \geq 0\}$

Vemos que $[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!] = \{(a^n, b^n c^n) \mid n \geq 0\}$, pero el lenguaje que define el ouput no es regular, y por lo tanto $[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!]$ no es racional. ■

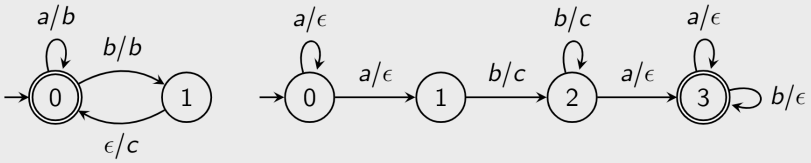
Definición. Decimos que un transductor $\mathcal{T}$ define una **función parcial** si:

para todo $u \in \Sigma^*$ se tiene que $|[\![\mathcal{T}]\!](u)| \leq 1$.

Definición. Decimos que $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ es **determinista** si cumple que:

1. $\mathcal{T}$ define una función $[\![\mathcal{T}]\!] : \Sigma^* \to \Omega^*$.
2. para todo $(p, a_1, b_1, q_1) \in \Delta$ y $(p, a_2, b_2, q_2) \in \Delta$, si $a_1 = a_2$, entonces
$$b_1 = b_2 \quad \text{y} \quad q_1 = q_2$$
Es decir, si tenemos dos transiciones desde un estado p a otros dos estados q_1 y q_2, y ambas transiciones leen lo mismo: $a_1 = a_2$, entonces lo que imprime el transductor y los estados deben ser iguales: $b_1 = b_2$ y $q_1 = q_2$. Básicamente, buscamos que sean la misma transición.
3. si $(p, \epsilon, b, q) \in \Delta$, entonces para todo $(p, a', b', q') \in \Delta$, se tiene que
$$(a', b', c') = (\epsilon, b, q)$$
Es decir, si tenemos una ϵ-transición, entonces es la única transición que sale desde p.

Ejemplo 3.3



3.3. Análisis léxico

Sección pendiente.

3.4. Algoritmo de Knuth-Morris-Prat

Pattern matching. Veamos el siguiente problema. Dado un **patrón** $w = w_1 \dots w_m$ y un documento $d = d_1 \dots d_n$, encontrar todas las posiciones donde aparece w en d, o sea, enumerar:

$$\{(i, j) \mid w = d_i d_{i+1} \dots d_j\}$$

Podríamos implementar la siguiente solución ingenua (y poco eficiente):

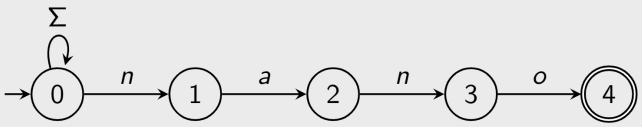
for $i = 0$ **to** $n - m$ **do**
  $j := 1$
  while $j \leq m \wedge w_j = d_{i+j}$ **do**
    $j := j + 1$
  if $j > m$ **then**
    output $(i + 1, i + m - 1)$

3.4.1. Autómata de un patrón

Definición. Dado una palabra $w = w_1 \dots w_m$, sea el NFA $\mathcal{A}_w = (Q, \Sigma, \Delta, I, F)$ tal que:

- $Q = \{0, 1, \dots, m\}$
- $\Delta = \{(0, a, 0) \mid a \in \Sigma\} \cup \{(i, w_{i+1}, i + 1) \mid i < m\}$
- $I = \{0\}$ y $F = \{m\}$.

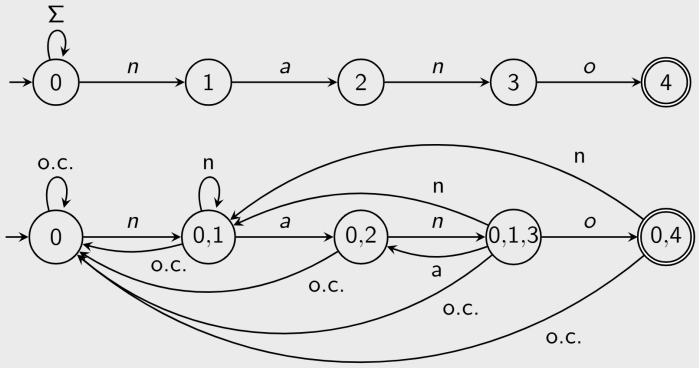
Ejemplo 3.4: Palabra $w =$ nano



Podemos usar $\mathcal{A}_w$ para encontrar todas las apariciones de w en w haciendo su **determinización**.

Defincíón. Sea $\mathcal{A}_w^{\text{det}} = \left(Q^{\text{det}}, \Sigma, \delta^{\text{det}}, \{0\}, F^{\text{det}}\right)$ la determinización de $\mathcal{A}_w$ tal que Q^{det} contiene **solo los estados alcanzables** desde $\{0\}$.

Ejemplo 3.5: Palabra $w =$ nano y $d =$ un nano no nana



q_0	q_0	q_1	q_0	q_1	q_2	q_3✓	q_4	q_0	q_1	q_0	q_0	q_1	q_2	q_3	q_2
u	n		n	a	n	o		n	o		n	a	n	a	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

Teorema 15

Para todo $S \in Q^{\text{det}}$ y $i \in \{0, 1, \ldots, m\}$ se cumple que

$$i \in S \quad \textit{si, y solo si,} \quad w_1 \ldots w_i \textit{ es un sufijo de } w_1 \ldots w_{\text{máx}(S)}.$$

Corolarios. Tenemos que:

- Para todo $S_1, S_2 \in Q^{\text{det}}$, si máx$(S_1) =$ máx(S_2), entonces $S_1 = S_2$.
- $\mathcal{A}_w^{\text{det}}$ tiene $|w| + 1$ estados y a lo más $\mathcal{O}(|w|^2)$ transiciones.

Por lo tanto, encontrar todos los substrings de w en d toma tiempo $\mathcal{O}(|d| + |w|^2)$.

Demostración teorema 15. Sea $S \in Q^{\text{det}}$ un conjunto de estados cualquiera alcanzable desde $\{0\}$. Entonces, existe una palabra $u = a_1 \ldots a_k$ tal que $\hat{\delta}^{\text{det}}(\{0\}, u) = S$. Por la demostración de que $\mathcal{L}\left(\mathcal{A}^{\text{det}}\right) = \mathcal{L}(\mathcal{A})$ para todo NFA $\mathcal{A}$, sabemos que $j \in S$ si, y sólo si, existe una ejecución $\mathcal{A}_w$ sobre u:

$$0 = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} q_k = j$$

Por la definición de $\mathcal{A}_w$ esta ejecución es de la forma:

$$0 \xrightarrow{a_1} 0 \xrightarrow{a_2} \dots \xrightarrow{a_{k-j}} 0 \underbrace{\xrightarrow{a_{k-j+1}} 1 \xrightarrow{a_{k-j+2}} 2 \dots \xrightarrow{a_k} j}_{w_1 \dots w_j}$$

Por lo tanto, $w_1 w_2 \dots w_j$ es sufijo de $a_1 \dots a_k$. Usaremos este último hecho para demostrar **ambas** direcciones del teorema, que formalizamos a continuación.

Propiedad. Para toda $u = a_1 \dots a_k$ tal que $\hat{\delta}^{\text{det}}(\{0\}, u) = S$, y para todo $j \leq m$:

$$j \in S \quad \text{si, y solo si,} \quad w_1 \dots w_j \text{ es sufijo de } a_1 \dots a_k$$

Demostración ($\Rightarrow$). Como S es alcanzable desde $\{0\}$, entonces existe $u = a_1 \dots a_k$ tal que $\hat{\delta}^{\text{det}}(\{0\}, u) = S$. Como $\text{máx}(S) \in S$, entonces $W_1 \dots W_{\text{máx}}(S)$ es sufijo de $a_1 \dots a_k$.

Suponga que $i \in S$. Entonces $w_1 \dots w_i$ es sufijo de $a_1 \dots a_k$. Como $i \leq \text{máx}(S)$, entonces:

$$a_1 a_2 \dots a_{k-\text{máx}(S)} \overbrace{a_{k-\text{máx}(S)+1} \dots a_{k-i} \underbrace{a_{k-i+1} \dots a_k}_{w_1 \dots w_i}}^{w_1 \dots w_{\text{máx}(S)}}$$

Por lo tanto, $w_1 \dots w_i$ es sufijo de $w_1 \dots w_{\text{máx}(S)}$.

Demostración ($\Leftarrow$). Como S es alcanzable desde $\{0\}$, entonces existe $u = a_1 \dots a_k$ tal que $\hat{\delta}^{\text{det}}(\{0\}, u) = S$. Como $\text{máx}(S) \in S$, entonces $w_1 \dots w_{\text{máx}(S)}$ es sufijo de $a_1 \dots a_k$.

Suponga que $w_1 \dots w_i$ es sufijo de $w_1 \dots w_{\text{máx}(S)}$. Como $w_1 \dots w_i$ es sufijo de $w_1 \dots w_{\text{máx}(S)}$ y $w_1 \dots w_{\text{máx}(S)}$ es sufijo de u, entonces $w_1 \dots w_i$ es sufijo de $u = a_1 \dots a_k$.

Por la "propiedad", concluimos que $i \in S$. ■

3.4.2. Autómata finito con k-lookahead

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

- $\Sigma_\bullet = \Sigma^* \times \Sigma^*$
- $\Sigma_\bullet^k = \{(u, v) \in \Sigma_\bullet \mid |uv| = k\}$

Notación. En vez de $(u, v) \in \Sigma_\bullet$, escribiremos $u.v \in \Sigma_\bullet$.

Ejemplo 3.6

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

- $ab.ba \in \Sigma_\bullet$ y $.aba \in \Sigma_\bullet$
- $ab.ba \in \Sigma_\bullet^4$ y $.aba \in \Sigma_\bullet^3$

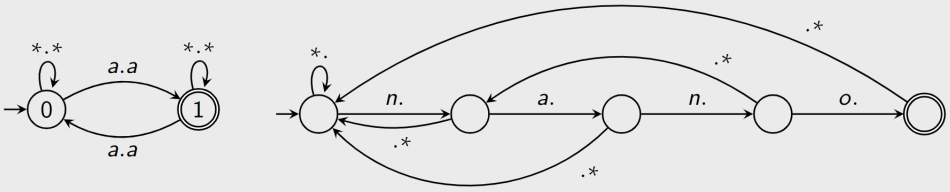
Definición. Un autómata finito determinista con k-lookahead es una tupla:

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

- Q es un conjunto finito de estados.
- Σ es el alfabeto del input.
- q_0 es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados iniciales.
- $\delta : Q \times (\Sigma \cup \{\$\})_\bullet^k \to Q$ es una función parcial, tal que:

$$\text{para todo } p \in Q \text{ y } w \in (\Sigma \cup \{\$\})^k : \quad |\{u.v \mid \delta(p, u.v) = q \text{ y } uv = w\}| \leq 1$$

Ejemplo 3.7



Ejecución. Sea $\mathcal{A}$ un DFA con k-lookahead. Tenemos que:

- Un par $(q, w) \in Q \times (\Sigma \cup \{\$\})^*$ es una **configuración** de $\mathcal{A}$.
- Una configuración $(q_0, w\$^k)$ es **inicial**.
- Una configuración $(q, \$^k)$ es **final** si $q \in F$.

El sufijo $\$^k$ nos sirve para marcar el final del input (y simplificar la definición de lookahead al leer el final de la palabra).

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

$$(p_1, w_1) \vdash_{\mathcal{A}} (p_2, w_2)$$

si, y sólo si, $\delta(p_1, u.v) = p_2$ y existe $w \in \Sigma^*$ tal que $w_1 = uvw$ y $w_2 = vw$.

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

$$\begin{array}{c} \text{para toda configuración } (p, w): \quad (p, w) \vdash^*_{\mathcal{A}} (p, w) \\ \text{si } (p_1, w_1) \vdash^*_{\mathcal{A}} \text{ y } (p2, w_2) \vdash_{\mathcal{A}} (p_3, w_3): \quad (p_1, w_1) \vdash^*_{\mathcal{A}} (p_3, w_3) \end{array}$$

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

Aceptación. Decimos que $\mathcal{A}$ **acepta** w si existe una configuración inicial $(q_0, w\$^k)$ y una configuración final $(q_f, \$^k)$ tal que:

$$\left(q_0, w\#^k\right) \vdash^*_{\mathcal{A}} \left(q_f, \#^k\right)$$

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

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

Vemos que son las mismas definiciones para un ϵ-NFA.

Teorema 16

*Para todo DFA con k-lookahead $\mathcal{A}$ se tiene que $\mathcal{L}(\mathcal{A})$ es un **lenguaje regular**.*

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

Definición. Llamaremos un **lazy autómata** a un DFA con 1-lookahead.

3.4.3. Algoritmo KMP

Construcción de un lazy automata. Sea $w = w_1 \dots w_m$ y $\mathcal{A}^{\text{det}}_w = \left(Q^{\text{det}}, \Sigma, \delta^{\text{det}}, \{0\}, F^{\text{det}}\right)$ la determinización de $\mathcal{A}_w$.

Definición. Para $i \in [0, m]$, sea S_i el **único estado** en Q^{det} tal que $i = \text{máx}(S_i)$.

Propiedad. Para todo $a \in \{w_1, \dots w_m\}$ y $i \in [0, m-1]$:

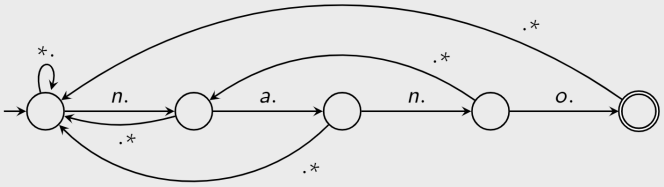
1. $S_i - \{i\} \in Q^{\text{det}}$
2. $a = w_{i+1}$, entonces $\delta^{\text{det}}(S_i, a) = S_{i+1}$.
3. $a \neq w_{i+1}$, entonces $\delta^{\text{det}}(S_i, a) = \delta^{\text{det}}(S_i - \{i\}, a)$.

La demostración de esta propiedad queda como ejercicio propuesto al lector.

En base a la propiedad anterior, se define el lazy autómata $\mathcal{A}_w^{\text{lazy}} = \left(Q^{\text{det}}, \Sigma, \delta^{\text{lazy}}, \{0\}, F^{\text{det}}\right)$ tal que:

- para todo $a \neq w_1$: $\delta^{\text{lazy}}(\{0\}, a.) = \{0\}$.
- para todo $a \in \{w_1, \dots, w_m\}$ y $i \in [0, m-1]$:
  - si $a = w_{i_1}$, entonces $\delta^{\text{lazy}}(S_i, a.) = S_{i+1}$
  - si $a \neq w_{i+1}$ y $i \neq 0$, entonces $\delta^{\text{lazy}}(S_i, .a) = S_i - \{i\}$.

Ejemplo 3.8



Teorema 17

Para todo w se cumple que $\mathcal{L}\left(\mathcal{A}_w^{det}\right) = \mathcal{L}\left(\mathcal{A}_w^{lazy}\right)$.

La demostración queda como ejercicio propuesto para el lector (usando la propiedad).

Complejidad KMP. Vemos que:

- El número de pasos que $\mathcal{A}_w^{\text{lazy}}$ **consume** letras $= |d|$
- El número de pasos que $\mathcal{A}_w^{\text{lazy}}$ **retrocede** $\leq |d|$
- El número de **pasos totales** de $\mathcal{A}_w^{\text{lazy}} \leq 2 \cdot |d|$

Por lo tanto, la cantidad de pasos es **lineal** en $\mathcal{O}(|d|)$.

Algoritmo KMP. Dado una palabra w y un documento d:

1. Construimos $\mathcal{A}_w^{\text{lazy}}$ desde $\mathcal{A}_w$.
2. Ejecutamos $\mathcal{A}_w^{\text{lazy}}$ sobre d.

El paso 1 toma $\mathcal{O}(|w|)$ y el paso 2 toma $\mathcal{O}(|d|)$, por lo tanto, el tiempo del algoritmo es $\mathcal{O}(|w| + |d|)$.

Queda como ejercicio para el lector demostrar que construir $\mathcal{A}_w^{\text{lazy}}$ toma tiempo $\mathcal{O}(|w|)$.

4. Lenguajes libres de contexto

4.1. Gramáticas libres de contexto

4.1.1. Gramáticas

Definición. Una **gramática libre de contexto** (CFG) es una tupla:

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

- V es un conjunto finito de **variables** o **no-terminales**.
- Σ es el alfabeto finito (o **terminales**) tal que $\Sigma \cap V = \varnothing$.
- $P \subseteq V \times (V \cup \Sigma)^*$ es un subconjunto finito de **reglas** o **producciones**.
- $S \in V$ es la **variable inicial**.

Ejemplo 4.1

Considere la gramática $\mathcal{G} = (V, \Sigma, P, S)$ tal que:

- $V = \{X, Y\}$
- $\Sigma = \{a, b\}$
- $\{(X, aXb), (X, Y), (Y, \epsilon)\}$
- $S = X$

$$\begin{aligned} \mathcal{G}: \quad & X \to aXb \\ & X \to Y \\ & Y \to \epsilon \end{aligned}$$

Notación. En este texto:

- Para las **variables** en una gramática usaremos letras mayúsculas: $X, Y, Z, A, B, C, \ldots$
- Para los **terminales** en una gramática usaremos letras minúsculas: $a, b, c, \ldots$
- Para palabras en $(V \cup \Sigma)^*$ usaremos símbolos: $\alpha, \beta, \gamma, \ldots$
- Para una producción $(A, \alpha) \in P$ la escribimos como: $A \to \alpha$

Simplificación. Si tenemos un conjunto de reglas de la forma:

$$\begin{aligned} X &\to \alpha_1 \\ X &\to \alpha_2 \\ &\cdots \\ X &\to \alpha_n \end{aligned}$$

entonces escribimos estas reglas **sucintamente** como

$$X \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$$

Recordando que $\alpha_1, \alpha_2, \ldots, \alpha_n \in (V \cup \Sigma)^*$.

Ejemplo 4.2

La gramática del ejemplo anterior:

$$\begin{aligned} \mathcal{G}: \quad & X \to aXb \\ & X \to Y \\ & Y \to \epsilon \end{aligned}$$

Podemos escribirla en notación **sucinta** como:

$$\begin{aligned} \mathcal{G}: \quad & X \to aXb \mid Y \\ & Y \to \epsilon \end{aligned}$$

Producciones. Sea $\mathcal{G}$ una CFG. Definimos la relación $\Rightarrow \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ de **producción** tal que:

$$\alpha \cdot X \cdot \beta \Rightarrow \alpha \cdot \gamma \cdot \beta \quad \text{si, y solo si,} \quad (X \to \gamma) \in P$$

para todo $X \in V$ y $\alpha, \beta, \gamma \in (V \cup \Sigma)^*$.

Si $\alpha X \beta \Rightarrow \alpha\gamma\beta$ entonces decimos que:

- $\alpha X \beta$ **produce** $\alpha\gamma\beta$ o
- $\alpha\gamma\beta$ **es producible** desde $\alpha X \beta$.
- $\alpha X \beta \Rightarrow \alpha\gamma\beta$ es **reemplazar** γ en X en la palabra $\alpha X \beta$.

Derivaciones. Sea $\mathcal{G}$ una CFG. Dadas dos palabras $\alpha, \beta \in (V \cup \Sigma)^*$ decimos que α **deriva** β:

$$\alpha \overset{*}{\Rightarrow} \beta$$

si existe $\alpha_1, \alpha_2, \ldots, \alpha_n \in (V \cup \Sigma)^*$ tal que: $\alpha \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \beta$, con $\overset{*}{\Rightarrow}$ la **clausura refleja y transitiva** de $\Rightarrow$, esto es:

1. $\alpha \overset{*}{\Rightarrow} \alpha$
2. $\alpha \overset{*}{\Rightarrow} \beta$ si, y sólo si, existe γ tal que $\alpha \overset{*}{\Rightarrow} \gamma$ y $\gamma \Rightarrow \beta$.

para todo $\alpha, \beta \in (V \cup \Sigma)^*$. Notemos que $\Rightarrow$ y $\overset{*}{\Rightarrow}$ son relaciones entre palabras en $(V \cup \Sigma)^*$.

Lenguaje. Sea $\mathcal{G}$ una CFG. El **lenguaje** de una gramática $\mathcal{G}$ se define como:

$$\mathcal{L}(\mathcal{G}) = \left\{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \right\}$$

$\mathcal{L}(\mathcal{G})$ son todas las palabras en Σ^* que se pueden derivar desde S.

Ejemplo 4.3

Sea $\mathcal{G}$ una CFG tal que:

$$\begin{aligned} \mathcal{G}: \quad & X \to aXb \mid Y \\ & Y \to \epsilon \end{aligned}$$

- Como $X \overset{*}{\Rightarrow} aaabbb$, entonces $aaabbb \in \mathcal{L}(\mathcal{G})$.
- En general, uno puede demostrar por **inducción** que:

$$\mathcal{L}(\mathcal{G}) = \{a^n b^n \mid n \geq 0\}$$

Lenguaje libre de contexto. Diremos que $L \subseteq \Sigma^*$ es un **lenguaje libre de contexto** si, y sólo si, existe una gramática libre de contexto $\mathcal{G}$ tal que:

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

Ejemplo 4.4

Los siguientes son lenguajes libres de contexto:

- $L = \{a^n b^n \mid n \geq 0\}$
- Par $= \{w \in \{a,b\}^* \mid w \text{ tiene largo par }\}$
- Pal $= \{w \in \{a,b\}^* \mid w = w^{\text{rev}}\}$

4.1.2. Árboles y derivaciones

Definición. El conjunto de **árboles ordenados y etiquetados** (o solo árboles) sobre etiquetas Σ y V, se define recursivamente como:

- $t := a$ es un árbol para todo $a \in \Sigma$.
- si $t_1, \ldots, t_k$ son árboles, entonces $t := X(t_1, \ldots, t_k)$ es un árbol para todo $X \in V$.

Para un árbol $t = X(t_1, \ldots, t_k)$ cualquiera se define:

- $\texttt{raiz}(t) = X$
- $\texttt{hijos}(t) = t_1, \ldots, t_k$

Si $t = a$, entonces decimos que t es una **hoja**, $\texttt{raiz}(t) = a$ y $\texttt{hijos}(t) = \epsilon$.

Definición. Fije una CFG $\mathcal{G} = (V, \Sigma, P, S)$. Se define el conjunto de **árboles de derivación** recursivamente como:

- Si $a \in \Sigma$, entonces $t = a$ es un árbol de derivación.
- Si $X \to X_1 \ldots X_k \in P$ y $t_1, \ldots, t_k$ son árboles de derivación con $\texttt{raiz}(t) = X_i$ para todo $i \leq k$, entonces $t = X(t_1, \ldots, t_k)$ es un árbol de derivación.

Decimos que t es un **árbol de derivación de** $\mathcal{G}$ si:

1. t es un árbol de derivación y

2. $\texttt{raiz}(t) = S$.

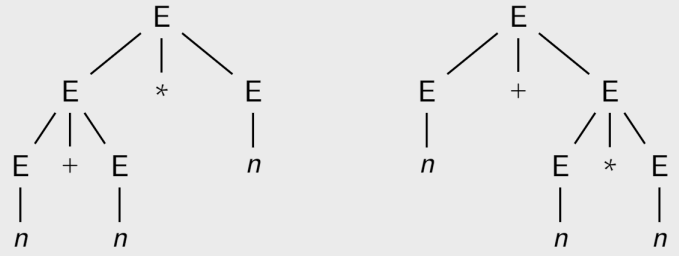
Los árboles de derivación son todos los árboles que parten desde S.

Ejemplo 4.5

Sea $\mathcal{G}$ una CFG tal que:

$$G: \quad E \rightarrow E + E \mid E * E \mid n$$

Algunos árboles de derivación para $\mathcal{G}$ son:



Definición. Sea $\mathcal{G}$ una CFG y $w \in \Sigma^*$. Se define la función `yield` sobre árboles, recursivamente como:

- Si $t = a \in \Sigma$, entonces $\texttt{yield}(t) = a$.
- Si t no es una hoja y $\texttt{hijos}(t) = t_1 t_2 \dots t_k$, entonces:

$$\texttt{yield}(t) = \texttt{yield}(t_1) \cdot \texttt{yield}(t_2) \cdot \ldots \cdot \texttt{yield}(t_k)$$

Decimos que t es un **árbol de derivación de $\mathcal{G}$ para** w si:

1. t es un árbol de derivación de $\mathcal{G}$ y
2. $\texttt{yield}(t) = w$

Lo anterior significa que las hojas de t forman la palabra w.

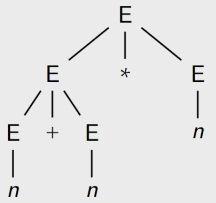
Proposición. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG y $w \in \Sigma^*$. Tenemos que:

$$w \in \mathcal{L}(\mathcal{G}) \quad \text{si, y solo si,} \quad \text{existe un árbol de derivación de } \mathcal{G} \text{ para } w.$$

Un árbol de derivación es la **representación gráfica** de una derivación.

Ejemplo 4.6

$$E \rightarrow E+E \mid E*E \mid n$$



1) $E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow n+E*E \Rightarrow n+n*E \Rightarrow n+n*n$
2) $E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow E+n*E \Rightarrow E+n*n \Rightarrow n+n*n$
3) $E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow E+E*n \Rightarrow E+n*n \Rightarrow n+n*n$
4) $E \Rightarrow E*E \Rightarrow E*n \Rightarrow E+E*n \Rightarrow n+E*n \Rightarrow n+n*n$
5) $E \Rightarrow E*E \Rightarrow E*n \Rightarrow E+E*n \Rightarrow E+n*n \Rightarrow n+n*n$
6) ...

Definición. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG.

- Definimos la **derivación por la izquierda** $\underset{\text{lm}}{\Rightarrow} \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$:

$$w \cdot X \cdot \beta \underset{\text{lm}}{\Rightarrow} w \cdot \gamma \cdot \beta \quad \text{si, y solo si,} \quad X \to \gamma \in P$$

  para todo $X \in V$, $w \in \Sigma^*$ y $\beta, \gamma \in (V \cup \Sigma)^*$.

- Definimos la **derivación por la derecha** $\underset{\text{rm}}{\Rightarrow} \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$:

$$\alpha \cdot X \cdot w \underset{\text{rm}}{\Rightarrow} \alpha \cdot \gamma \cdot w \quad \text{si, y solo si,} \quad X \to \gamma \in P$$

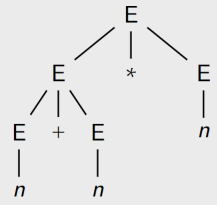
  para todo $X \in V$, $w \in \Sigma^*$ y $\alpha, \gamma \in (V \cup \Sigma)^*$.

Se define $\overset{*}{\underset{\text{lm}}{\Rightarrow}}$ y $\overset{*}{\underset{\text{rm}}{\Rightarrow}}$ como la **clausura refleja y transitiva** de $\underset{\text{lm}}{\Rightarrow}$ y $\underset{\text{rm}}{\Rightarrow}$, respectivamente.

$\underset{\text{lm}}{\Rightarrow}$ y $\underset{\text{rm}}{\Rightarrow}$ solo reemplaza **a la izquierda** (leftmost) y **derecha** (rightmost).

Ejemplo 4.7

$$E \rightarrow E+E \mid E*E \mid n$$



Derivación por la izquierda (lm)

$$E \underset{\text{lm}}{\Rightarrow} E*E \underset{\text{lm}}{\Rightarrow} E+E*E \underset{\text{lm}}{\Rightarrow} n+E*E \underset{\text{lm}}{\Rightarrow} n+n*E \underset{\text{lm}}{\Rightarrow} n+n*n$$

Derivación por la derecha (rm)

$$E \underset{\text{rm}}{\Rightarrow} E*E \underset{\text{rm}}{\Rightarrow} E*n \underset{\text{rm}}{\Rightarrow} E+E*n \underset{\text{rm}}{\Rightarrow} E+n*n \underset{\text{rm}}{\Rightarrow} n+n*n$$

Proposición. Por cada árbol de derivación, existe una **única** derivación por la izquierda y una **única** derivación por la derecha.

Por lo tanto, desde ahora podemos hablar de **árbol de derivación y derivación (izquierda o derecha)** indistintamente.

4.1.3. Lenguajes regulares vs libres de contexto

Proposición. Para todo lenguaje regular L, existe una gramática libre de contexto $\mathcal{G}_\mathcal{A}$:

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

Idea demostración. Dado un autómata finito determinista $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, ¿cómo construimos una gramática libre de contexto?

Defina la gramática $\mathcal{G}_\mathcal{A} = (Q, \Sigma, P_\mathcal{A}, q_0)$ tal que:

- si $\delta(p, a) = q$, entonces $p \to aq \in P_\mathcal{A}$
- si $p \in F$, entonces $p \to \epsilon \in P_\mathcal{A}$

Queda como ejercicio propuesto al lector demostrar que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{G}_\mathcal{A})$

4.2. Simplificación de gramáticas

¿Cómo podemos simplificar la siguiente gramática?

$$\begin{aligned} G: \quad S &\to aAa \mid aBD \mid aBH \\ A &\to B \mid D \\ B &\to aBa \mid b \\ C &\to aCC \mid bC \\ D &\to aDCa \mid CFa \\ F &\to aFDa \mid aab \\ H &\to \epsilon \end{aligned}$$

1. Dada una variable X, ¿es X **útil** para producir palabras?
2. Dada una producción $p : X \to \gamma$, ¿es p **útil** para producir palabras?

4.2.1. Eliminación de variables inútiles

Definición. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG. Diremos que una variable $X \in V$ es **útil** si existe una derivación:

$$\boxed{S \overset{\star}{\Rightarrow} \alpha X \beta \overset{\star}{\Rightarrow} w}$$

Al contrario, diremos que una variable X es **inútil** si NO es útil.

Definición. Para una variable $X \in V$:

1. Decimos que X es **alcanzable** si existe una derivación:

$$\boxed{S \overset{\star}{\Rightarrow} \alpha X \beta}$$

2. Decimos que X es **generadora** si existe una derivación:

$$\boxed{X \overset{\star}{\Rightarrow} w}$$

Propiedad. Para toda variable $X \in V - \{S\}$:

existe una producción $Y \to \alpha X \beta \in P$ tal que $Y \in V$ es alcanzable $\quad \Leftrightarrow \quad$ X es alcanzable.

La demostración de esta propiedad queda como ejercicio propuesto al lector.

Un algoritmo para determinar si una variable es alcanzable:

input : Gramática $\mathcal{G} = (V, \Sigma, P, S)$
output: Conjunto C de variables alcanzables
Function `alcanzables(`$\mathcal{G}$`):`
- **let** $C_0 := \{S\}$
- **let** $C := \varnothing$
- **while** $C_0 \neq \varnothing$ **do**
  - **take** $Y \in C_0$
  - $C_0 := C_0 - \{Y\}$
  - $C := C \cup \{Y\}$
  - **foreach** $X \in V - C$ *tal que existe una regla* $\Big(Y \to \alpha X \beta\Big) \in P$ **do**
    - $C_0 := C_0 \cup \{X\}$
- **return** C

Propiedad. Para toda variable $X \in V$:

existe una regla $X \to \alpha$ tal que todas las variables en α son generadoras $\quad \Leftrightarrow \quad$ X es generadora.

La demostración de esta propiedad queda como ejercicio propuesto al lector.

Un algoritmo para determinar si una variable es generadora:

input : Gramática $\mathcal{G} = (V, \Sigma, P, S)$
output: Conjunto G de variables generadoras
Function `alcanzables(`$\mathcal{G}$`):`
- **let** $G_0 := \{X \in V \mid (X \to w) \in P\}$
- **let** $G := \varnothing$
- **while** $G_0 \neq G$ **do**
  - $G := G_0$
  - **foreach** $(X \to \alpha) \in P$ **do**
    - **if** *todas las variables en* α *estan en* G **then**
      - $G_0 := G_0 \cup \{X\}$
- **return** G

Teorema 18

Sea $G = (V, \Sigma, P, S)$ una CFG. Sea $\mathcal{G}''$ una gramática creada a partir de $\mathcal{G}$ después de:

- *eliminar todas las variables y reglas NO generadoras.*
- *eliminar todas las variables y reglas NO alcanzables.*

Entonces, $\mathcal{L}(\mathcal{G}'') = \mathcal{L}(\mathcal{G})$ y $\mathcal{G}''$ no contiene variables inútiles.

Nota: Debemos respetar el orden para eliminar variables generadoras y luego las alcanzables. De hacerlo al revés, la gramática resultante puede no definir el mismo lenguaje que la inicial.

Demostración teorema 18. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG.

Sea $\mathcal{G}' = (V', \Sigma, P', S)$ al eliminar las variables **no generadoras** de $\mathcal{G}$:

$$V' = \{X \in V \mid \exists w.\ X \overset{\star}{\underset{\mathcal{G}}{\Rightarrow}} w\}$$
$$P' = \left\{X \to \alpha \in P \mid X \in V' \wedge \alpha \in (V' \cup \Sigma)^*\right\}$$

Sea $\mathcal{G}'' = (V'', \Sigma, P'', S)$ al eliminar las variables **no alcanzables** de $\mathcal{G}'$:

$$V'' = \left\{X \in V' \mid \exists \alpha, \beta.\ S \overset{\star}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha X \beta\right\}$$
$$P'' = \left\{X \to \alpha \in P' \mid X \in V'' \wedge \alpha \in (V'' \cup \Sigma)^*\right\}$$

Considere las siguientes propiedades de $\mathcal{G}$, $\mathcal{G}'$ y $\mathcal{G}''$:

1. Para todo $\alpha \in (V \cup \Sigma)^*$, si $\alpha \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} w$ entonces $\alpha \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} w$.
2. Para todo $\alpha \in (V' \cup \Sigma)^*$, si $S \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha$ entonces $S \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} \alpha$
3. Para todo $\alpha \in (V'' \cup \Sigma)^*$, si $\alpha \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} w$ entonces $\alpha \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} w$.

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

Demostración $\mathcal{L}(\mathcal{G}'') \subseteq \mathcal{L}(\mathcal{G})$. Como $V'' \subseteq V$ y $P'' \subseteq P$, entonces es trivial que $\mathcal{L}(\mathcal{G}'') \subseteq \mathcal{L}(\mathcal{G})$.

Demostración $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{G}'')$. Sea $w \in \mathcal{L}(\mathcal{G})$ tal que $S \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} w$.

- Por la propiedad 1, tenemos que $S \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} w$.
- Por la propiedad 2, tenemos que $S \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} w$.

Por lo tanto $w \in \mathcal{L}(\mathcal{G}'')$ y concluimos que $\mathcal{L}(\mathcal{G}'') \subseteq \mathcal{L}(\mathcal{G})$.

Demostración variables útiles. Queremos mostrar que para todo $X \in V''$, X es **útil** en $\mathcal{G}''$.

Como $X \in V''$, entonces $S \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha X \beta$ para algún $\alpha, \beta \in (V' \cup \Sigma)^*$.

Por la propiedad 2, se tiene que: $S \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} \alpha X \beta$ y $\alpha, \beta \in (V'' \cup \Sigma)^*$.

Como $X \in V'$ y $\alpha, \beta \in (V' \cup \Sigma)^*$, entonces existen u, v, w tal que:

$$\alpha \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} u, \quad X \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} v, \quad \beta \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} w$$

Por la propiedad 1, se tiene que: $\alpha \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} u, \quad X \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} v, \quad \beta \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} w$.

Por la propiedad 3, se tiene que: $\alpha \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} u, \quad X \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} v, \quad \beta \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} w$.

Juntando todo, $S \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} \alpha X \beta \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} uvw$ y por tanto X es útil en $\mathcal{G}''$. ■

4.2.2. Eliminación de producciones inútiles

Definición. Sea $\mathcal{G}$ una CFG. Decimos que:

- Una producción de la forma $X \to \epsilon$ es **en vacío**.
- Una producción de la forma $X \to Y$ es **unitaria**.

Deseamos eliminar este tipo de producciones para simplificar nuestras gramáticas, sin embargo, debemos tener cuidado con algunos detalles.

Proposición. Si $\epsilon \in \mathcal{L}(\mathcal{G})$, entonces NO se pueden borrar las producciones en vacío sin alterar el lenguaje $\mathcal{G}$.

Así que, desde ahora, supondremos que $\epsilon \notin \mathcal{L}(\mathcal{G})$.

Definición. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG tal que $\epsilon \notin \mathcal{L}(\mathcal{G})$. Definimos a P^* como el **menor conjunto de producciones** que contiene a P y **cerrado bajo** las siguientes reglas:

1. Si $X \to Y \in P^*$ y $Y \to \gamma \in P^*$, entonces $X \to \gamma \in P^*$.

2. Si $X \to \epsilon \in P^*$ y $Z \to \alpha X \beta \in P^*$, entonces $Z \to \alpha\beta \in P^*$.

Definimos $\mathcal{G}^* = (V, \Sigma, P^*, S)$. Entonces:

- P^* es finito y
- $\mathcal{L}(\mathcal{G}^*) = \mathcal{L}(\mathcal{G})$.

Ahora, para cualquier palabra $w \in \mathcal{L}(\mathcal{G}^*)$, sea $\mathcal{T}$ un árbol de derivación de w en $\mathcal{G}^*$ de **tamaño mínimo**. Definimos las siguientes propiedades:

1. El árbol de derivación $\mathcal{T}$ NO usa una **producción unitaria**.

2. El árbol de derivación $\mathcal{T}$ NO usa una **producción en vacío**.

La demostración de estas propiedades se hace por contradicción: se supone que $\mathcal{T}$ usa una producción unitaria o en vacío y se comprueba que si ocurre esto entonces $\mathcal{T}$ no tiene tamaño mínimo.

Por la propiedad 1 y 2, tenemos que *para todo $w \in \mathcal{L}(\mathcal{G}^*)$, existe una derivación de w en $\mathcal{G}$ que NO usa producciones **en vacío** ni producciones **unitarias***. Por lo tanto, podemos eliminar las producciones en vacío y unitarias de $\mathcal{G}^*$.

Teorema 19

Para toda CFG $\mathcal{G}$ tal que $\epsilon \notin \mathcal{L}(\mathcal{G})$, sea:

- *$\mathcal{G}^*$ la clausura de producciones unitarias y en vacío.*
- *$\hat{\mathcal{G}}$ el resultado de remover toda producción unitaria o en vacío de $\mathcal{G}^*$.*

Entonces, $\mathcal{L}(\hat{\mathcal{G}}) = \mathcal{L}(\mathcal{G})$ y $\hat{\mathcal{G}}$ no tiene producciones unitarias o en vacío.

Resumiendo, para eliminar las producciones en vacío y unitarias de $\mathcal{G}$:

- construimos $\mathcal{G}^*$ haciendo la **clausura** de producciones unitarias y en vacío,
- construimos $\hat{\mathcal{G}}$ **removiendo** todas las producciones unitarias o en vacío de $\mathcal{G}^*$.

Por el resultado anterior sabemos que $\mathcal{L}(\hat{\mathcal{G}}) = \mathcal{L}(\mathcal{G})$. Es importante mencionar que es posible que $\hat{\mathcal{G}}$ contenga símbolos inútiles.

4.3. Forma normal de Chomsky

Definición. Una gramática $\mathcal{G}$ esta en **forma normal de Chomsky** (CNF) si todas sus reglas son de la forma:

- $X \to YZ$
- $X \to a$

Ejemplo 4.8

La siguiente gramática está en CNF:

$$\begin{aligned} S &\to AB \mid AC \mid SS \\ C &\to SB \\ A &\to a \\ B &\to b \end{aligned}$$

Toda gramática se puede convertir en CNF. Para facilitar el proceso, considere $\mathcal{G} = (V, \Sigma, P, S)$ una CFG tal que $\epsilon \notin \mathcal{L}(\mathcal{G})$.

- Primero, suponga que $\mathcal{G}$ no contiene reglas en vacío o unitarias.
- Por lo tanto, todas las reglas en $\mathcal{G}$ son de la forma:
  - $X \to \gamma \quad$ para $|\gamma| \geq 2$
  - $X \to a$

Los pasos para convertir $\mathcal{G}$ en CNF son:

1. Convertir todas las reglas a la forma:
   - $X \to Y_1 Y_2 \dots Y_k$ para $k \geq 2$
   - $X \to a$
2. Convertir todas las reglas a la forma:
   - $X \to YZ$
   - $X \to a$

Para realizar el paso 1:

- Para cada $a \in \Sigma$, agregamos una nueva variable X_a y una regla $X_a \to a$.
- Reemplazamos todas las ocurrencias antiguas de a por X_a.

Ejemplo 4.9

Considere la gramática $S \to aSb \mid ab$, entonces, hacer el paso 1 termina en:

$$\begin{aligned} S &\to ASB \mid AB \\ A &\to a \\ B &\to b \end{aligned}$$

Correctitud. Si $\mathcal{G}'$ es la gramática resultante del paso 1, entonces se cumple que $\mathcal{L}(\mathcal{G}') = \mathcal{L}(\mathcal{G})$.

Para realizar el paso 2, tomamos cada regla $p : X \to Y_1 Y_2 \dots Y_k$ con $k \geq 3$ y:

- Agregamos una **nueva** variable Z.
- Reemplazamos la regla p por **dos reglas**:

$$X \to Y_1 Z \quad \text{y} \quad Z \to Y_2 \dots Y_k$$

Repetimos este paso hasta llegar a la forma normal de Chosmky.

Ejemplo 4.10

El resultado del paso 1 anterior es:

$$\begin{aligned} S &\to ASB \mid AB \\ A &\to a \\ B &\to b \end{aligned}$$

Al realizar el paso 2, la gramática queda de la forma:

$$\begin{aligned} S &\to AZ \mid AB \\ Z &\to SB \\ A &\to a \\ B &\to b \end{aligned}$$

Correctitud. Si $\mathcal{G}''$ es la gramática resultante del paso 2, entonces se cumple que $\mathcal{L}(\mathcal{G}'') = \mathcal{L}(\mathcal{G}')$.

Teorema 20

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG tal que $\epsilon \notin \mathcal{L}(\mathcal{G})$. Existe una gramática $\mathcal{G}'$ en forma normal de Chomsky tal que:

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

Si $\mathcal{G}'$ no tiene reglas unitarias ni en vacío, entonces $\mathcal{G}'$ es de **tamaño polinomial** con respecto a $\mathcal{G}$.

4.4. Lema de bombeo para lenguajes libres de contexto

Similarmente para los lenguajes regulares, existe un lema de bombeo para lenguajes libres de contexto.

Sea $L \subseteq \Sigma^*$. Si L es **libre de contexto**, entonces:

(LB^{CFL}) existe un $N > 0$ tal que
para toda palabra $z \in L$ con $|z| \geq N$
existe una descomposición $z = uvwxy$
con $vx \neq \epsilon$ y $|vwx| \leq N$ tal que
para todo $i \geq 0$, $u \cdot v^i \cdot w \cdot x^i \cdot y \in L$

Análogamente, para demostrar que un lenguaje L NO es libre de contexto, usamos el contrapositivo del lema.

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

($\neg\text{LB}^{\text{CFL}}$) para todo $N > 0$ tal que
existe una palabra $z \in L$ con $|z| \geq N$
para toda descomposición $z = uvwxy$
con $vx \neq \epsilon$ y $|vwx| \leq N$ tal que
existe $i \geq 0$, $u \cdot v^i \cdot w \cdot x^i \cdot y \notin L$
entonces L NO es libre de contexto.

Jugando contra un demonio. El lema de bombeo puede verse como el siguiente "juego":




"L NO es CFL" "L es CFL"

El escoge un $N > 0$

Uno escoge $z \in L$ **con** $|z| \geq N$

El escoge $u\,v\,w\,x\,y = z$ **con** $vx \neq \epsilon$ **y** $|vwx| \leq N$

Uno escoge $i \geq 0$

Uno gana si $u \cdot v^i \cdot w \cdot x^i \cdot y \notin L$ **El gana** si $u \cdot v^i \cdot w \cdot x^i \cdot y \in L$

Ejemplo 4.11

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




"a^{n^2} NO es CFL" "a^{n^2} es CFL"

Escojo $N > 0$

Yo escojo $a^{N^2} \in L$

Entonces escojo $\underbrace{a^j}_{u}\underbrace{a^k}_{v}\underbrace{a^l}_{w}\underbrace{a^m}_{x}\underbrace{a^n}_{y} = a^{N^2}$

con $k + m \neq 0$ **y** $k + l + m \leq N$

Yo escojo $i = 2$

Ganamos ya que al bombear con $i = 2$ ya no se cumple que $j + k + l + m + n = N^2$ (se rompe el equilibrio), y entonces $z \notin L$.

Ejemplo 4.12

Considere el lenguaje $L = \{a^n b^n c^n \mid n > 0\}$.





"$a^n b^n c^n$ NO es CFL" "$a^n b^n c^n$ es CFL"

Escojo $N > 0$

Yo escojo $a^N b^N c^N \in L$

Entonces escojo $uvwxy = a^N b^N c^N$ **con** $vx \neq \epsilon$ **y** $|vwx| \leq N$

Yo escojo $i = 2$

Ganamos el juego, ya que, como $uvwxy = a^N b^N c^N$ con $vx \neq \epsilon$ y $|vwx| \leq N$, entonces

$$vwx \in \mathcal{L}(a^* b^*) \quad \text{o} \quad vwx \in \mathcal{L}(b^* c^*)$$

- Si $vwx \in \mathcal{L}(a^+ b^+)$, entonces:
  - $|uv^2wx^2y|_{a,b} > 2N$
  - $|uv^2wx^2y|_c = N$

  Por lo tanto, $z' \notin L$.
- Si $vwx \in \mathcal{L}(b^+ c^+)$, entonces:
  - $|uv^2wx^2y|_{b,c} > 2N$
  - $|uv^2wx^2y|_a = N$

  Por lo tanto, $z' \notin L$.

En ambos casos, $uv^2wx^2y \notin L$, y por tanto L no es CFG.

Lema versión juego. *"Dado un lenguaje $L \subseteq \Sigma^*$, si **UNO** tiene una estrategia ganadora en el juego $(\neg LB^{CFL})$ para toda estrategia posible del demonio, entonces L **NO** es libre de contexto".*

Consecuencias. Podemos establecer la siguiente proposición en base al lema de bombeo:

- Para todos lenguajes libres de contexto L_1 y L_2, se cumple que $L_1 \cup L_2$ es un lenguaje libre de contexto.
- Existen lenguajes libres de contexto L, L_1 y L_2 tales que:
  - $L_1 \cap L_2$ **NO** es un lenguaje libre de contexto.
  - L^C **NO** es un lenguaje libre de contexto.

Para demostrar que la intersección no es libre de contexto, basta con un contraejemplo. Tomemos $L_1 = \{a^n b^n c^m \mid n \geq 0, m \geq 0\}$ y $L_2 = \{a^m b^n c^n \mid n \geq 0, m \geq 0\}$, al hacer su intersección, obtendremos el lenguaje $L = \{a^n b^n c^n \mid n \geq 0\}$, que ya vimos que no es libre de contexto.

Queda como ejercicio propuesto para el lector demostrar que L^C no es libre de contexto.

4.5. Algoritmo CKY

Dado un lenguaje libre de contexto L y una palabra w, ¿cómo verificamos si $w \in L$?

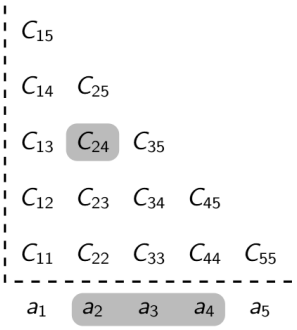
- Convertimos $\mathcal{G}$ en forma normal de Chomsky.
- Probamos todas las derivaciones de altura a lo más $|w| + 1$.
- Si encontramos una derivación retornamos `TRUE`.

Los pasos anteriores se conoce como el algoritmo CKY, inventado por John Cocke, Tadao Kasami y Daniel Younger. Es un algoritmo que usa **programación dinámica**, y es **cúbico** en $|w|$ y **lineal** en $|\mathcal{G}|$:

Tiempo: $\mathcal{O}(|w|^3 \cdot |\mathcal{G}|)$

Por simplicidad, asumiremos que las gramáticas que recibe el algoritmo están en **Forma Normal de Chomsky** (CNF).

Tabla del algoritmo CKY. Para una palabra $w = a_1 a_2 \dots a_n$ y una gramática $\mathcal{G} = (V, \Sigma, P, S)$ construimos la **tabla CKY**:

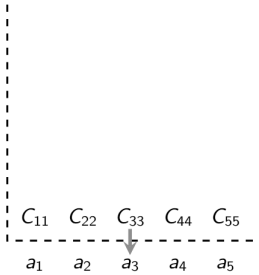


Para todo $1 \leq 1 \leq j \leq n$ se define:

$$C_{i\,j} = \{X \in V \mid X \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} a_i \dots a_j\}$$

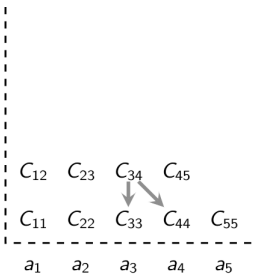
Paso 0 (inicial). Para cada i, construimos el conjunto $C_{ii} \subseteq V$ tal que:

$$C_{i\,i} = \{X \in V \mid X \to a_i \in P\}$$



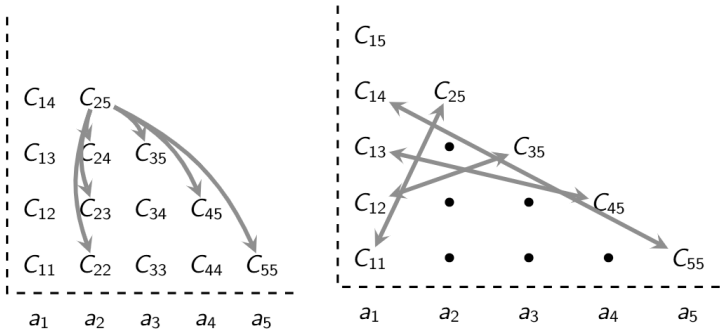
Paso 1. Para cada i, construimos el conjunto $C_{i\,i+1} \subseteq V$ tal que:

$$C_{i\,i+1} = \{X \in V \mid X \to YZ \in P \text{ para algún } Y \in C_{ii} \wedge Z \in C_{i+1i+1}\}$$



Paso k $(k > 0)$**.** Para cada i, construimos el conjunto $C_{i\,i+k} \subseteq V$ tal que:

$$C_{i\,i+k} = \{X \in V \mid \exists j \in [i, i+k].\ X \to YZ \in P \text{ para algún } Y \in C_{i\,j} \wedge Z \in C_{j+i\,i+k}\}$$



Ejemplo 4.13

Considere la palabra *bbaba* y la gramática:

S	→	XY | YZ
X	→	YY | a
Y	→	YX | b
Z	→	XZ | XX | a

$\{S, Y\}$				
$\{S, Y\}$	$\{X, Z\}$			
$\{X, Z\}$	$\{X\}$	$\{S\}$		
$\{X\}$	$\{S, Y\}$	$\{S\}$	$\{S, Y\}$	
$\{Y\}$	$\{Y\}$	$\{X, Z\}$	$\{Y\}$	$\{X, Z\}$
b	b	a	b	a

Considere la palabra *abba* y la gramática:

S	→	XY | YX | SS | XZ | YW
X	→	a
Y	→	b
Z	→	SY
W	→	SX

$\{S\}$			
$\{Z\}$	$\varnothing$		
$\{S\}$	$\varnothing$	$\{S\}$	
$\{X\}$	$\{Y\}$	$\{Y\}$	$\{X\}$
a	b	b	a

Algoritmo CKY. A continuación se muestra el pseudo-código del algoritmo:

input : Una gramática $\mathcal{G} = (V, \Sigma, P, S)$ y una palabra $w = a_1 a_2 \dots a_n$

output: `TRUE` si, y sólo si, $w \in \mathcal{L}(\mathcal{G})$

Function `AlgoritmoCKY`$(\mathcal{G}, w)$**:**

- **for** $i = 1$ **to** n **do**
  - **let** $C_{i\,i} = \varnothing$
  - **for** $X \to C \in P$ **do**
    - **if** $c = a_i$ **then**
      - **let** $C_{i\,i} = C_{i\,i} \cup \{X\}$
- **for** $k = 1$ **to** $n - 1$ **do**
  - **for** $i = 1$ **to** $n - k$ **do**
    - **let** $C_{i\,i+k} = \varnothing$
    - **for** $j = i$ **to** $i + k - 1$ **do**
      - **for** $X \to YZ \in P$ **do**
        - **if** $Y \in C_{i\,j} \wedge Z \in C_{j+1\,i+k}$ **then**
          - **let** $C_{i\,i+k} = C_{i\,i+k} \cup \{X\}$
- **return check** $S \in C_{1\,n}$

Análisis algoritmo CKY. En su correctitud, para toda gramática $\mathcal{G}$ y para toda palabra $w \in \Sigma^*$ se tiene que:

$$\texttt{AlgoritmoCKY}(\mathcal{G}, w) = \texttt{TRUE} \quad \Leftrightarrow \quad w \in \mathcal{L}(\mathcal{G})$$

La demostración queda como ejercicio propuesto al lector.

Si el input es de tamaño $|w|$ y la gramática es de tamaño $|\mathcal{G}|$, entonces:

$$\text{Tiempo del algoritmo CKY:} \quad \mathcal{O}(|w|^3 \cdot |\mathcal{G}|)$$

5. Algoritmos para lenguajes libres de contexto

5.1. Autómatas apiladores

5.1.1. Versión normal

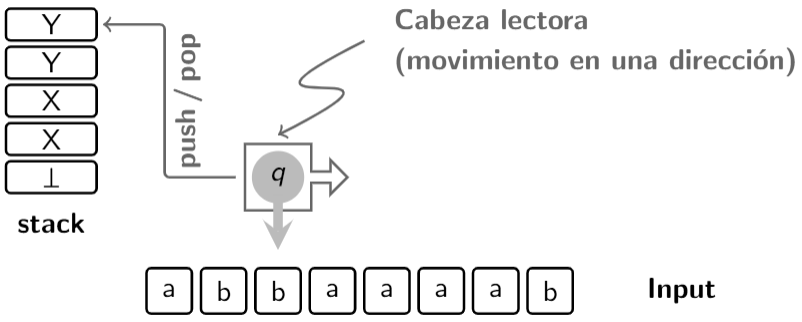


Figura 8: Idea de un autómata apilador

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

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

- 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**.
- $\bot \in \Gamma$ 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:

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

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**:

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

si el autómata apilador está:

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

entonces:

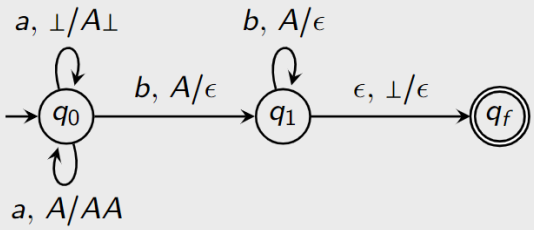
♦ cambia al estado q, y modifico el tope A por $B_1 B_2 \cdots B_k$.

Ejemplo 5.1

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

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

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



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 $q \cdot \gamma = q_0 \cdot \bot$.

♦ 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 $\mathcal{P}$:

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

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

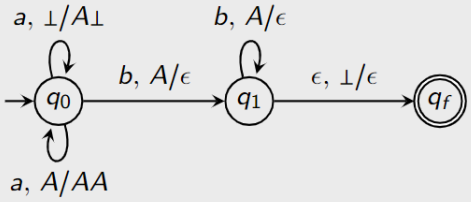
- $w_1 = a \cdot w_2$
- $\gamma_1 = A \cdot \gamma$
- $\gamma_2 = \alpha \cdot \gamma$

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:



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

Definiciones. $\mathcal{P}$ **acepta** w si, y sólo si, $(q_0 \perp, w) \vdash^*_{\mathcal{P}} (q_f, \epsilon)$ 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 Riveros le gustó y lo encontró interesante.

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

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

- 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:

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

Ejemplo 5.4

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

- $Q = \{\bot, q_0, q_1, q_f\}$ y Δ:

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

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

Teorema 21

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	Autómatas apiladores
Terminales (Σ)	Alfabeto input (Σ)
Variables (V)	Alfabeto stack (Γ)
Producciones	Transiciones
$A \to \gamma$	$pA \xrightarrow{c} q\gamma$
$A \to a$	$pA \xrightarrow{c} q$
Derivaciones	Ejecuciones
Árbol de derivación	Stack

Figura 9: Gramáticas vs Autómatas apiladores

Teorema 22

*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 23

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:

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

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 $\gamma \in Q^+$:

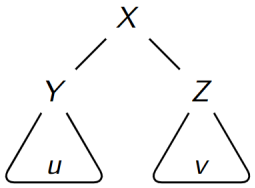
$$(X \cdot \gamma, a) \vdash_{\mathcal{D}} (a \cdot \gamma, a) \vdash_{\mathcal{D}} (\gamma, \epsilon)$$

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 24

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

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

donde $\vdash^n_{\mathcal{P}}$ 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

$$(pA, \underbrace{auv}_{w}) \vdash^n_{\mathcal{P}} (p', \epsilon) \text{ ssi } (pA, auv) \vdash_{\mathcal{P}} (p_1A_1A_2, uv) \vdash^i_{\mathcal{P}} (p_2A_2, v) \vdash^j_{\mathcal{P}} (p', \epsilon)$$

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

■

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

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

- $V = \Gamma$.
- 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.

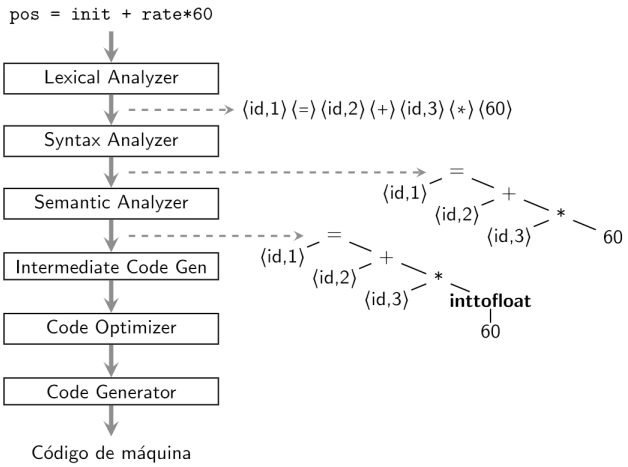


Figura 10: La estructura de un compilador

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

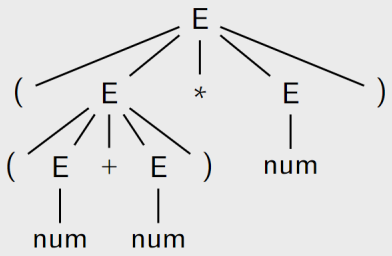
$$E \ \rightarrow \ (E + E) \mid (E * E) \mid \text{num}$$

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:

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

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 $X \rightarrow \alpha \mid \beta$, ¿cómo elegir entre α o β?

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

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

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

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

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 $w = a_1 a_2 \dots a_n \in \Sigma^*$ se define el k-**prefijo** de w como:

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

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

$$u \odot_k v = (u \cdot v)|_k$$

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:

$$L|_k = \{w|_k \mid w \in L\}$$
$$L_1 \odot_k L_2 = \{w_1 \odot_k w_2 \mid w_1 \in L_1 \text{ y } w_2 \in L_2\}$$

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 $\gamma \in (V \cup \Sigma)^*$:

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

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:

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

Ejemplo 5.10

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

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

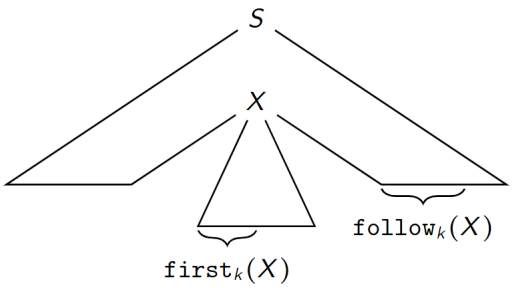


Figura 11: Representación de `first` y `follow`

5.3.3. Calcular First

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

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

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

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

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

■

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 $X \in (V \cup \Sigma)$:

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

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 25

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 $X \overset{*}{\Rightarrow} w$, 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**
    $\texttt{first}_k^i(X) := \bigcup_{X \to X_1 \dots X_n \in P} \texttt{first}_k^{i-1}(X_1) \odot_k \cdots \odot_k \texttt{first}_k^{i-1}(X_n)$
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 $X \neq S$:

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

Si consideramos $X = S$:

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

Dado lo anterior, podemos definir el siguiente teorema.

Teorema 26

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

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

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

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

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

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 $i \leq k \cdot |\Sigma|^k \cdot |V|$:

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

Teorema 27

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:

- $S \overset{*}{\underset{\text{lm}}{\Rightarrow}} uY\beta \underset{\text{lm}}{\Rightarrow} u\gamma_1\beta \overset{*}{\underset{\text{lm}}{\Rightarrow}} uv_1$
- $S \overset{*}{\underset{\text{lm}}{\Rightarrow}} uY\beta \underset{\text{lm}}{\Rightarrow} u\gamma_2\beta \overset{*}{\underset{\text{lm}}{\Rightarrow}} uv_2$ 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)

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

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

- 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).

$$\mathcal{G}_2: \quad S \to (X) \mid n$$
$$X \to SX \mid \epsilon$$

$$S \overset{\star}{\underset{\text{lm}}{\Rightarrow}} uX\beta \underset{\text{lm}}{\Rightarrow} u\gamma_1\beta \overset{\star}{\underset{\text{lm}}{\Rightarrow}} uv_1$$
$$\underset{\text{lm}}{\Rightarrow} u\gamma_2\beta \overset{\star}{\underset{\text{lm}}{\Rightarrow}} uv_2$$

- 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)

$$\begin{aligned} \mathcal{G}_3: \quad S &\to (X) \mid n+S \mid n \\ X &\to SX \mid \epsilon \end{aligned}$$

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

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)

$$\begin{aligned} \mathcal{G}_4: \quad S &\to (X) \mid (X)\hat{}e \mid n+S \mid n \\ X &\to SX \mid \epsilon \end{aligned}$$

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

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:

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

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

Ejemplo 5.15: Lenguaje NO LL(k)

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

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

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 28

$\mathcal{G}$ es una gramática LL(k) si, y sólo 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:

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

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

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

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:

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

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 $Y \to \gamma_1, Y \to \gamma_2 \in P$ se tiene que:

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

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 $S \overset{*}{\underset{\text{lm}}{\Rightarrow}} uY\beta$, se tiene que

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

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

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 $X \to \epsilon$:

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

¿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:

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

Por lo tanto, establecemos el siguiente teorema.

Teorema 29

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:

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

Teorema 30

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

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

■

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

S	$\to$	$Xa \mid b$		S	$\to$	$Xa \mid b$
X	$\to$	$Xc \mid d$		X	$\to$	dX'
				X'	$\to$	$cX' \mid \epsilon$

Teorema 31

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 $X_i \to X_j\gamma \in P'$ **do**
        foreach $X_j \to \alpha \in P'$ **do**
          $P' := P' \cup \{X_i \to \alpha\gamma\}$
        $P' := P' - \{X_i \to X_j\gamma\}$
    Remover recursión inmediata para X_i en P' (si existe)
  $V' := \{X_1, \ldots, X_n\} \cup \{X'_1, \ldots, X'_n\}$
  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)

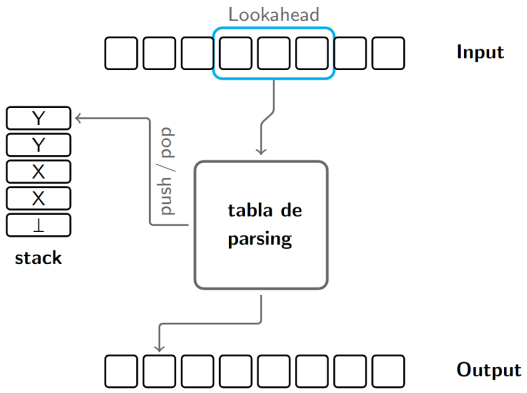


Figura 12: Máquina de parsing

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

$$\boxed{\alpha_1 \text{ NO es prefijo de } \alpha_2 \quad \text{o} \quad u_1v_1 \text{ NO es prefijo de } u_2v_2.}$$

*"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 $\mathcal{G}$:

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

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 $u \in \texttt{first}_k(\gamma) \odot_k \texttt{follow}_k(X)$

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 LL(k) 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 $\mathcal{T}[\mathcal{G}]$ 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 $M[X, u] \in (V \cup \Sigma)^* \cup \{\texttt{ERROR}\}$:

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

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 $M[x, a] \in (V \cup \Sigma)^* \cup \{\texttt{ERROR}\}$:

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

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

Imaginemos que tenemos un log de la forma

```
18:30   ERROR 06
19:10   OK 00
20:00   ERROR 19
```

y queremos obtener todas las horas `HH:MM`, quizá con una expresión regular $R = (\backslash d\backslash d:\backslash d\backslash d)$. ¿Cómo podemos **automatizar** esta tarea de extraer datos?

1. Una tarea como la anterior **se puede programar fácilmente**, ¿por qué queremos automatizarla?
2. Las expresiones regulares ya hacen el trabajo anterior, ¿por qué queremos **estudiar este problema de nuevo**?

En esta última sección veremos **nuevas técnicas formales y algorítmicas** para entender y resolver este problema.

6.1.1. Spans

Definición. Para un documento $d = a_0 a_1 \dots a_{n-1}$ se define un **span** s de d como:

$$\boxed{s = [i, j\rangle}$$

tal que $0 \leq i \leq j \leq n$.

Si $s = [i, j\rangle$ es un span de d se define:

$$\boxed{d[s] = d[[i, j\rangle] = a_i a_{i+1} \dots a_{j-1}}$$

como el **contenido del span** s en d. Si $i = j$, entonces $d[[i, i\rangle] = \epsilon$.

Ejemplo 6.1



$s_1 = [15, 20\rangle \quad w[s_1] = 19{:}10 \qquad s_2 = [32, 32\rangle \quad w[s_2] = \epsilon$

Denotamos el conjunto de **todos los spans** de d como $\texttt{Spans}(d)$.

6.1.2. Regex

Sintaxis. Sea Σ un alfabeto finito y $\mathcal{X}$ un conjunto de variables. R es una expresión regular con variables (o **regex**) sobre Σ si R es igual a:

1. a, para alguna letra $a \in \Sigma$.
2. ϵ
3. $\mathbf{x}\{R_1\}$, donde R_1 es una regex y $\mathbf{x} \in X$.
4. $(R_1 + R_2)$, donde R_1 y R_2 son regex.
5. $(R_1 \cdot R_2)$, donde R_1 y R_2 son regex.
6. (R_1^*), donde R_1 es una regex.

$\mathbf{x}\{R_1\}$ representa que el **span capturado** por R_1 lo almacenaremos en la variable $\mathbf{x}$.

Ejemplo 6.2

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

- $\Sigma^* \cdot \mathbf{x}\{abba\} \cdot \Sigma^*$
- $\Sigma^* \cdot \sqcup \cdot \mathbf{x}\{ab \cdot (a+b)^*\} \cdot \sqcup \cdot \Sigma^*$
- $\Sigma^* \cdot \sqcup \cdot \mathbf{x}\{(a+b)^*\} \cdot \sqcup \cdot \mathbf{y}\{(a+b)^*\} \cdot \sqcup \cdot \Sigma^*$
- $(\Sigma^* \cdot \sqcup + \epsilon) \cdot \mathbf{z}\{\mathbf{x}\{(a+b)^+\} \cdot \sqcup \cdot \mathbf{y}\{(a+b)^+\}\} \cdot (\epsilon + \sqcup \cdot \Sigma^*)$
- $\Sigma^* \cdot \mathbf{x}\{abba\} \cdot \sqcup \cdot \mathbf{x}\{baba\} \cdot \Sigma^*$

Regex válidas. Sea $\texttt{Var}(R)$ el conjunto de **todas las variables** en $\mathcal{X}$ mencionadas en R. Decimos que R es una regex **válida** si, y sólo si, todas las siguientes condiciones se cumplen:

1. si $R = R_1 \cdot R_2$, entonces $\texttt{Var}(R_1) \cap \texttt{Var}(R_2) = \varnothing$, y R_1 y R_2 son válidas.
2. si $R = R_1 + R_2$, entonces $\texttt{Var}(R_1) = \texttt{Var}(R_2)$, y R_1 y R_2 son válidas.
3. si $R = R_1^*$, entonces $\texttt{Var}(R_1) = \varnothing$, y R_1 es válida.
4. si $R = \mathbf{x}\{R_1\}$, entonces $\mathbf{x} \notin \texttt{Var}(R_1)$, y R_1 es válida.

Si R es **válida** nos permite asignar las variables de **manera correcta**.

Ejemplo 6.3

Las siguientes regex son válidas:

- $\Sigma^* \cdot \mathbf{x}\{abba\} \cdot \Sigma^*$
- $\Sigma^* \cdot \mathbf{x}\{abba\} \cdot \sqcup \cdot \mathbf{y}\{baba\} \cdot \Sigma^*$
- $\Sigma^* \cdot \sqcup \cdot \mathbf{x}\{ab \cdot (a+b)^*\} \cdot \sqcup \cdot \Sigma^*$

Las siguientes regex **no** son válidas:

- $\Sigma^* \cdot \mathbf{x}\{abba\} \cdot \sqcup \cdot \mathbf{x}\{baba\} \cdot \Sigma^*$
- $\Sigma^* \cdot (\mathbf{x}\{abba\} + \mathbf{y}\{baba\}) \cdot \Sigma^*$
- $\Sigma^* \cdot \sqcup \cdot (\mathbf{x}\{ab \cdot (a+b)^*\} \cdot \sqcup)^* \cdot \Sigma^*$

Mappings. Un **mapping** de R sobre un documento $d \in \Sigma^*$ es una función:

$$\boxed{\mu : \texttt{Var}(R) \to \texttt{Spans}(d)}$$

donde el dominio de μ es $\texttt{dom}(\mu) = \texttt{Var}(R)$.

Se define el mapping $\mu = \perp$ como el **mapping vacío** donde $\texttt{dom}(\perp) = \varnothing$.

Ejemplo 6.4



$$[\mathbf{x} \mapsto [0,5\rangle, \mathbf{y} \mapsto [6,11\rangle, \mathbf{z} \mapsto [12,14\rangle]$$

$$[\mathbf{x} \mapsto [15,20\rangle, \mathbf{y} \mapsto [21,23\rangle, \mathbf{z} \mapsto [24,26\rangle]$$

$$\perp = [\cdot]$$

- Para una variable $\mathbf{x}$ y span s se define el **mapping** de una variable:

$$\mu = [\mathbf{x} \mapsto s] \quad \text{tal que} \quad \texttt{dom}(\mu) = \{\mathbf{x}\} \quad \text{y} \quad \mu(\mathbf{x}) = s$$

- Para $k \in \mathbb{N}$ se define el **mapping** $\mu + k$ tal que $\texttt{dom}(\mu + k) = \texttt{dom}(\mu)$ y:

$$\text{si } \mu(\mathbf{x}) = [i, j\rangle \text{ entonces } [\mu + k](\mathbf{x}) = [i + k, j + k\rangle.$$

- Para mappings μ_1, μ_2 con $\texttt{dom}(\mu_1) \cap \texttt{dom}(\mu_2) = \varnothing$ se define la **unión**:

$$\mu = \mu_1 \cup \mu_2 \quad \text{tal que} \quad \mu(\mathbf{x}) = \begin{cases} \mu_1(\mathbf{x}) & \text{si } \mathbf{x} \in \text{dom}\,(\mu_1) \\ \mu_2(\mathbf{x}) & \text{si } \mathbf{x} \in \text{dom}\,(\mu_2) \end{cases}$$

Semántica regex. Para una regex válida R cualquiera, se define la función $[\![R]\!]$ **inductivamente** sobre documentos $d \in \Sigma^*$:

1. $[\![a]\!](d) = \{\perp\}$ si $d = a$, y $\varnothing$ en otro caso.
2. $[\![\epsilon]\!](d) = \{\perp\}$ si $d = \epsilon$, y $\varnothing$ en otro caso.
3. $[\![\mathbf{x}\,\{R_1\}]\!](d) = \{\mu \cup [\mathbf{x} \mapsto s] \mid \mu \in [\![R_1]\!](d) \text{ y } s = [0, |d|\rangle\}$
4. $[\![R_1 \cdot R_2]\!](d) = \left\{ \mu_1 \cup (\mu_2 + |d_1|) \;\middle|\; \begin{array}{c} \text{existe } d_1, d_2 \text{ tal que } d = d_1 \cdot d_2, \\ \mu_1 \in [\![R_1]\!](d_1) \text{ y } \mu_2 \in [\![R_2]\!](d_2) \end{array} \right\}$
   Como $R_1 \cdot R_2$ son válidas, entonces sabemos que $\texttt{dom}(\mu_1) \cap \texttt{dom}(\mu_2) = \varnothing$.
5. $[\![R_1 + R_2]\!](d) = [\![R_1]\!](d) \cup [\![R_2]\!](d)$
6. $[\![R_1^*]\!](d) = \bigcup_{k=0}^{\infty} [\![(R_1)^k]\!](d)$
   Como $\texttt{Var}(R_1) = \varnothing$, entonces $[\![R_1^*]\!](d) = \{\perp\}$ si, y sólo si, $d \in \mathcal{L}(R_1^*)$.

Ejemplo 6.5

- $[\![b]\!](a) = \varnothing$
- $[\![b]\!](b) = \{\perp\}$
- $[\![\mathbf{x}\{b\}]\!](a) = \varnothing$
- $[\![\mathbf{x}\{b\}]\!](b) = \{[\mathbf{x} \mapsto [0, 1\rangle]\}$
- $[\![\mathbf{x}\{b\}b]\!](bb) = \{[\mathbf{x} \mapsto [0, 1\rangle]\}$
- $[\![b\mathbf{x}\{b\}]\!](bb) = \{[\mathbf{x} \mapsto [1, 2\rangle]\}$
- $[\![\mathbf{x}\{b\}\mathbf{y}\{b\}]\!](bb) = \{[\mathbf{x} \mapsto [0, 1\rangle, \mathbf{y} \mapsto [1, 2\rangle]\}$
- $[\![\mathbf{x}\{b\}b + b\mathbf{x}\{b\}]\!](bb) = \{[\mathbf{x} \mapsto [0, 1\rangle], [\mathbf{x} \mapsto [1, 2\rangle]\}$

Ejemplo 6.6

$$d = \underset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16}{\texttt{a abba ba baba ba}}$$

- $[\![\Sigma^* \cdot \mathbf{x}\{aba + bab\} \cdot \Sigma^*]\!](d) = \{[\mathbf{x} \mapsto [10, 13\rangle], [\mathbf{x} \mapsto [11, 14\rangle]\}$
- $[\![\Sigma^* \cdot \textvisiblespace \cdot \mathbf{x}\{ba \cdot (a + b)^*\} \cdot \textvisiblespace \cdot \Sigma^*]\!](d) = \{[\mathbf{x} \mapsto [7, 9\rangle], [\mathbf{x} \mapsto [10, 14\rangle]\}$
- $[\![\Sigma^* \cdot \mathbf{x}\{abba\} \cdot \textvisiblespace \cdot \mathbf{y}\{ba\} \cdot \Sigma^*]\!](d) = \{[\mathbf{x} \mapsto [2, 6\rangle, \mathbf{y} \mapsto [7, 9\rangle]\}$
- $[\![\Sigma^* \cdot \textvisiblespace \cdot \mathbf{x}\{(a + b)^*\} \cdot \textvisiblespace \cdot \mathbf{y}\{(a + b)^*\} \cdot \textvisiblespace \cdot \Sigma^*]\!](d) = \{[\mathbf{x} \mapsto [2, 6\rangle, \mathbf{y} \mapsto [7, 9\rangle], [\mathbf{x} \mapsto [7, 9\rangle, \mathbf{y} \mapsto [10, 14\rangle]\}$
- $[\![\Sigma^* \cdot \mathbf{x}\{\Sigma^*\} \cdot \Sigma^*]\!](d) = \texttt{Spans}(d)$

Evaluación regex. Podemos preguntarnos:

- Dado una regex R con una variable, ¿de qué tamaño puede ser $|[\![R]\!](d)|$ con respecto a $|d|$ **en el peor caso**?
- Dado una regex R con k variables, ¿de qué tamaño puede ser $|[\![R]\!](d)|$ con respecto a $|d|$ y k **en el peor caso**?

La respuesta es que el tamaño de $|[\![R]\!](d)|$ puede crecer de manera exponencial. ¿Cómo podemos evaluar entonces una regex de R **eficientemente**? Analizamos una herramienta para lograr esto a continuación.

6.1.3. Vset autómata

¿Qué tiene de nuevo un autómata con variables (vset autómata)?

1. Tiene transiciones con **abre** y **cierra** de variable **x**:

$$p \xrightarrow{\langle x} q \quad y \quad p \xrightarrow{x\rangle} q$$

2. Cada ejecución define un **mapping** de las variables a spans.

Un **vset autómata** será nuestro primer modelo para **compilar regex**.

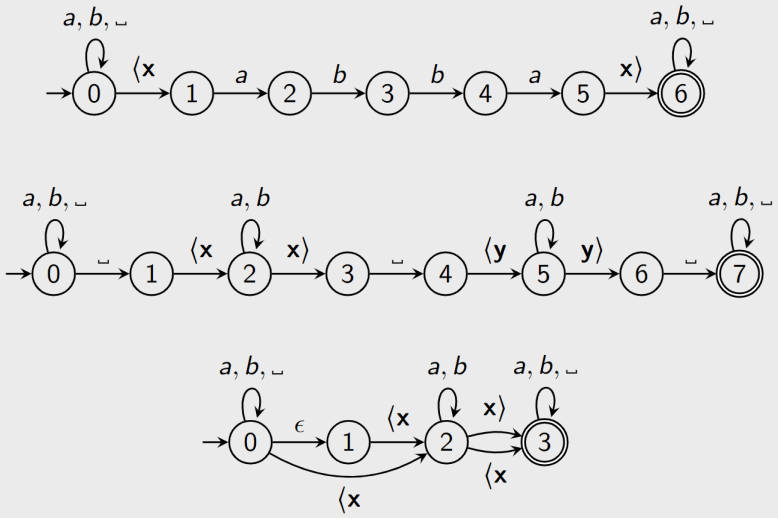
Definición. Un vset autómata (VA) es una tupla:

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

- Q es un conjunto finito de estados.
- Σ es el alfabeto de input.
- $\mathcal{X}$ es un conjunto finito de variables.
- $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\} \cup \{\langle \mathbf{x}, \mathbf{x}\rangle \mid \mathbf{x} \in \mathcal{X}\}) \times Q$ es la relación de transición.
- $I \subseteq Q$ es un conjunto de estados iniciales.
- $F \subseteq Q$ es el conjunto de estados finales (o aceptación).

'$\langle\mathbf{x}$' simboliza **abrir** y '$\mathbf{x}\rangle$' simboliza **cerrar** la variable $\mathbf{x}$.

Ejemplo 6.7: vset autómatas



Ejecución. Sea $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \Delta, I, F)$ un VA y $d = a_0 \dots a_{n-1} \in \Sigma^*$ un documento. Tenemos que:

- Un par $(p, i) \in Q \times \{0, \dots, n\}$ es una **configuración** de $\mathcal{A}$ sobre d.
- Una configuración $(p, 0)$ es **inicial** si $q \in I$.
- Una configuración $(p, |d|)$ es **final** si $q \in F$.

"Intuitivamente, una configuración (p, i) representa que $\mathcal{A}$ se encuentra en el estado p antes de leer a_i".

Una **ejecución** (o *run*) ρ de $\mathcal{A}$ sobre d es una secuencia:

$$\rho : (p_0, i_0) \xrightarrow{o_1} (p_1, i_1) \xrightarrow{o_2} \dots \xrightarrow{o_m} (p_m, i_m)$$

tal que cumple todas las siguientes condiciones:

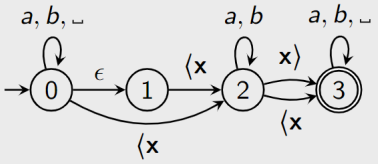
- $o_k \in \Sigma \cup \{\epsilon\} \cup \{\langle \mathbf{x}, \mathbf{x}\rangle \mid \mathbf{x} \in \mathcal{X}\}$ con $k \leq m$,

- (p_0, i_0) es una configuración inicial,
- para todo $k < m$, $(p_k, o_{k+1}, p_{k+1}) \in \Delta$,
- para todo $k \leq m$, si $o_k \in \Sigma$, entonces $o_k = a_{i_{k-1}}$ y $i_k = i_{k-1} + 1$,
- para todo $k \leq m$, si $o_k \in \{\epsilon\} \cup \{\langle \mathbf{x}, \mathbf{x}\rangle \mid \mathbf{x} \in \mathcal{X}\}$, entonces $i_k = i_{k-1}$.

Una ejecución ρ es de **aceptación** si (p_m, i_m) es de aceptación.

Ejemplo 6.8: Ejecuciones

$d = \underset{0\ 1\ 2}{\mathtt{aba}}$



Algunas ejecuciones sobre d:

- $\rho_1 : (p_0, 0) \xrightarrow{a} (p_0, 1) \xrightarrow{b} (p_0, 2) \xrightarrow{a} (p_0, 3)$
- $\rho_2 : (p_0, 0) \xrightarrow{a} (p_0, 1) \xrightarrow{\epsilon} (p_1, 1) \xrightarrow{\langle x} (p_2, 1) \xrightarrow{b} (p_2, 2) \xrightarrow{x\rangle} (p_3, 2) \xrightarrow{a} (p_3, 3)$
- $\rho_3 : (p_0, 0) \xrightarrow{a} (p_0, 1) \xrightarrow{\langle x} (p_2, 1) \xrightarrow{b} (p_2, 2) \xrightarrow{x\rangle} (p_3, 2) \xrightarrow{a} (p_3, 3)$
- $\rho_4 : (p_0, 0) \xrightarrow{a} (p_0, 1) \xrightarrow{\langle x} (p_2, 1) \xrightarrow{b} (p_2, 2) \xrightarrow{\langle x} (p_3, 2) \xrightarrow{a} (p_3, 3)$

Donde ρ_2 y ρ_3 son ejecuciones válidas.

Una ejecución ρ es **válida** si, y sólo si, para todo $\mathbf{x} \in \mathcal{X}$:

- existe un único $k_1 \leq m$ tal que $o_{k_1} = \langle \mathbf{x}$,
- existe un único $k_2 \leq m$ tal que $o_{k_2} = \mathbf{x}\rangle$ y
- $k_1 < k_2$.

"Intuitivamente, ρ es válida si, y sólo si, todas las variables se abren y cierran correctamente."

Si ρ es **válida** se define el **mapping de** ρ $\mathtt{map}(\rho) : \mathcal{X} \to \mathtt{Spans}(d)$ tal que:

$$\boxed{[\mathtt{map}(\rho)](\mathbf{x}) = [i_{k_1}, i_{k_2}\rangle}$$

para todo $\mathbf{x} \in \mathcal{X}$ y $k_1, k_2 \leq m$ con $i_{k_1} = \langle \mathbf{x}$ y $i_{k_2} = \mathbf{x}\rangle$.

Ejemplo 6.9: Mappings de ρ

El mapping para las ejecuciones válidas del ejemplo anterior es:

$$\mathtt{map}\,(\rho_2) = \mathtt{map}\,(\rho_3) = [\mathbf{x} \mapsto [1, 2\rangle]$$

Función de extracción. Sea $\mathcal{A}$ un VA. Se define la función $[\![\mathcal{A}]\!]$ tal que para todo documento $d \in \Sigma^*$:

$$[\![\mathcal{A}]\!](d) = \{\texttt{map}(d) \mid \rho \text{ es una ejecución válida y de aceptación de } \mathcal{A} \text{ sobre } d\}$$

Un VA nos entrega otra forma de extraer información de un documento.

6.1.4. Desde regex a VA

Definición. Sea $\mathcal{A}$ un VA. Decimos que $\mathcal{A}$ es **funcional** si, y sólo si, para todo documento d y para toda ejecución ρ de $\mathcal{A}$ sobre d:

si ρ es de aceptación, entonces ρ es válida.

Para funcional solo necesitamos verificar que la ejecución es de aceptación.

Teorema 32

*Para toda regex R válida, existe un vset autómata **funcional** $\mathcal{A}_R$ de **tamaño lineal** en $|R|$ tal que para todo documento d:*

$$[\![R]\!](d) = [\![\mathcal{A}_{\mathcal{R}}]\!](d)$$

La demostración de este teorema queda como ejercicio propuesto al lector (es similar al Teorema de Kleene).

6.2. Enumeración de resultados: Autómatas con anotaciones

Veamos el siguiente problema:

`PROBLEMA:`	Evaluación de regex
`INPUT:`	Una regex R y
	un documento w
`OUTPUT:`	Enumerar todos los mappings en $[\![R]\!](d)$

La idea es:

1. Transformamos R a un vset autómata $\mathcal{A}_R$.
2. Enumeramos los resultados $[\![\mathcal{A}_R]\!](d)$.

¿Cómo computamos todos los mappings en $[\![\mathcal{A}_R]\!](d)$? ¿Cómo los encontramos si son demasiados? Veamos qué podemos hacer.

6.2.1. Representación de mappings

Sea $d = a_0 \dots a_{n-1}$, un conjunto de variables $\mathcal{X}$ y un mapping $\mu : \mathcal{X} \to \texttt{Spans}(d)$.

Definiciones. Tenemos que:

1. Se define el **conjunto de marcas de** $\mathcal{X}$ como:

$$\texttt{Markers}(\mathcal{X}) = \{\langle \mathbf{x} \mid \mathbf{x} \in \mathcal{X}\} \cup \{\mathbf{x}\rangle \mid \mathbf{x} \in \mathcal{X}\}$$

2. Se define el **mapping inverso** de μ como $\mu^{\text{inv}} : [0, n] \to 2^{\text{Markers}(\mathcal{X})}$:

$$\mu^{\text{inv}}(i) = \{\langle \mathbf{x} \mid \exists j.\ \mu(\mathbf{x}) = [i, j\rangle \in \mathcal{X}\} \cup \{\mathbf{x}\rangle \mid \exists j.\ \mu(\mathbf{x}) = [j, i\rangle \in \mathcal{X}\}$$

3. Se define la **secuenciación** de μ como $\mathtt{seq}(\mu) = \mathtt{seq}_0(\mu) \cdot \ldots \cdot \mathtt{seq}_n(\mu)$:

$$\mathtt{seq}_i(\mu) = \begin{cases} \left(i, \mu^{\text{inv}}(i)\right) & \mu^{\text{inv}}(i) \neq \varnothing \\ \epsilon & \mu^{\text{inv}}(i) = \varnothing \end{cases}$$

$\mathtt{seq}(\mu)$ es una **representación equivalente** de un mapping μ, y nos será más conveniente para nuestros algoritmos de enumeración de resultados.

Ejemplo 6.10



$\mu(\mathbf{x}) = [15, 20\rangle$ $\quad \mu^{\text{inv}}(15) = \{\langle \mathbf{x}, \langle \mathbf{z}\}$ $\quad \mu^{\text{inv}}(20) = \{\mathbf{x}\rangle\}$

$\mu(\mathbf{y}) = [24, 26\rangle$ $\quad \mu^{\text{inv}}(24) = \{\langle \mathbf{y}\}$ $\quad \mu^{\text{inv}}(26) = \{\mathbf{y}\rangle, \mathbf{z}\rangle\}$

$\mu(\mathbf{z}) = [15, 26\rangle$ $\quad \forall i \notin \{15, 20, 24, 26\}$ $\quad \mu^{\text{inv}}(i) = \varnothing$

$$\mathtt{seq}(\mu) = (15, \{\langle \mathbf{x}, \langle \mathbf{z}\}) \ (20, \{\mathbf{x}\rangle\}) \ (24, \{\langle \mathbf{y}\}) \ (26, \{\mathbf{y}\rangle, \mathbf{z}\rangle\})$$

6.2.2. Autómatas con anotaciones

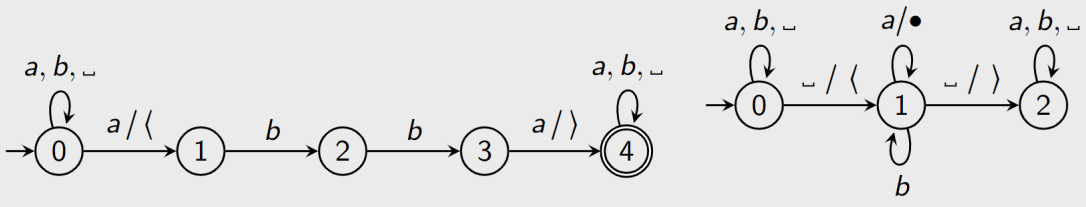
Definición. Un autómata con anotaciones (AnnA) es una tupla:

$$\mathcal{N} = (Q, \Sigma, \Lambda, \Delta, I, F)$$

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

Las transiciones (p, a, l, q) simbolizan que al leer la letra a, esta letra **será anotada** con l.

Ejemplo 6.11



Ejecución. Sea un AnnA $\mathcal{N} = (Q, \Sigma, \Lambda, \Delta, I, F)$ y un documento $d = a_0 \dots a_{n-1} \in \Sigma^*$. Una **ejecución** ρ de $\mathcal{N}$ sobre d es una secuencia:

$$\rho : p_0 \xrightarrow{t_0} p_1 \xrightarrow{t_1} \dots \xrightarrow{t_{n-1}} p_n$$

tal que cumple todas las siguientes condiciones:

1. $p_0 \in I$
2. para todo $i < n$, t_i es de la forma $t_i = a_i$ o $t_i = (a_i, l)$ para algún $l \in \Lambda$
3. para todo $i < n$, $(p_i, t_i, p_{i+1}) \in \Delta$

Se define la **anotación de** ρ como $\texttt{ann}(\rho) = \texttt{ann}_0(t_0) \cdot \ldots \cdot \texttt{ann}_n(t_{n-1})$:

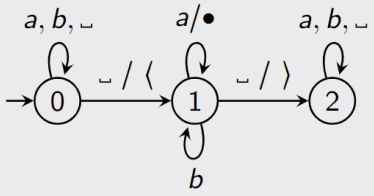
$$\texttt{ann}_i(t) = \begin{cases} (i, l) & t = (a, l) \\ \epsilon & t = a \end{cases}$$

Decimos que ρ es de **aceptación** si, y sólo si, $q_n \in F$.

Ejemplo 6.12: Ejecuciones

$d =$ a␣abba␣ba␣b

0 1 2 3 4 5 6 7 8 9 10



Algunas ejecuciones sobre d:

- $\rho_1 : p_0 \xrightarrow{a} p_0 \xrightarrow{\text{␣}/\langle} p_1 \xrightarrow{a/\bullet} p_1 \xrightarrow{b} p_1 \xrightarrow{b} p_1 \xrightarrow{a/\bullet} p_1 \xrightarrow{\text{␣}/\rangle} p_2 \xrightarrow{b} p_2 \xrightarrow{a} p_2 \xrightarrow{\text{␣}} p_2 \xrightarrow{b} p_2$
- $\rho_2 : p_0 \xrightarrow{a} p_0 \xrightarrow{\text{␣}} p_0 \xrightarrow{a} p_0 \xrightarrow{b} p_0 \xrightarrow{b} p_0 \xrightarrow{a} p_0 \xrightarrow{\text{␣}/\langle} p_1 \xrightarrow{b} p_1 \xrightarrow{a/\bullet} p_1 \xrightarrow{\text{␣}/\rangle} p_2 \xrightarrow{b} p_2$

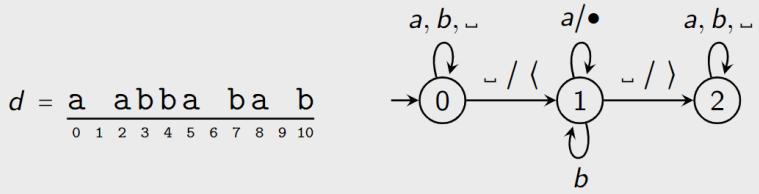
Además, tenemos que:

- $\texttt{ann}(\rho_1) = (1, \langle)(2, \bullet)(5, \bullet)(6, \rangle)$ $\quad a\,\langle\text{␣}\,\bullet a\,b\,b\,\bullet a\,\rangle\text{␣}\,b\,a\,\text{␣}\,b$
- $\texttt{ann}(\rho_2) = (6, \langle)(8, \bullet)(9, \rangle)$ $\quad a\,\text{␣}\,a\,b\,b\,a\,\langle\text{␣}\,b\,\bullet a\,\rangle\text{␣}\,b$

Output de un AnnA. Sea $\mathcal{N}$ un AnnA. Se define la función $[\![\mathcal{N}]\!]$ tal que para todo documento $d \in \Sigma^*$:

$$[\![\mathcal{N}]\!](d) = \{\texttt{ann}(\rho) \mid \rho \text{ es una ejecución aceptación de } \mathcal{N} \text{ sobre } d\}$$

Ejemplo 6.13: Output de un AnnA



Para el documento d se tiene que:

$$[\![\mathcal{N}]\!](d) = \{(1, \langle)(2, \bullet)(5, \bullet)(6, \rangle), (6, \langle)(8, \bullet)(9, \rangle)\}$$

6.2.3. Desde un vset a AnnA

Teorema 33

Sea Σ un alfabeto finito. Para todo vset autómata funcional $\mathcal{A}$ sobre Σ, existe un AnnA $\mathcal{N}$ sobre $\Sigma \cup \{\#\}$ tal que para todo documento d sobre Σ:

$$[\![\mathcal{N}]\!](d \cdot \#) = \{\text{seq}(\mu) \mid \mu \in [\![\mathcal{A}]\!](d)\}$$

El teorema anterior nos dice que $\mathcal{N}$ entrega la **secuenciación** de los mappings en $[\![\mathcal{A}]\!](d)$.

Propiedades. Sea $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \Delta, I, F)$ un vset autómata funcional y $p, q \in Q$. **Sin pérdida de generalidad**, desde ahora supondremos que todos los estados de un vset autómata funcional $\mathcal{A}$ son **útiles**. En otras palabras, para todo estado $p \in Q$ de $\mathcal{A}$:

- Existe una ejecución (camino de transiciones) desde I a p.
- Existe una ejecución (camino de transiciones) desde p a F.

Definición. Una **ejecución sin lectura** (ejec-SL) de p a q en $\mathcal{A}$ es una secuencia:

$$\pi : \quad p_0 \xrightarrow{s_0} p_1 \xrightarrow{s_1} \ldots \xrightarrow{s_{k-1}} p_k$$

donde:

- $p_0 = p$ y $p_k = q$
- para todo $i < k$, (p_i, s_i, p_{i+1}) y $s_i \in \texttt{Markers}(\mathcal{X}) \cup \{\epsilon\}$.

Una ejecución sin lectura es un **camino de transiciones** de p a q tal que $s_i \notin \Sigma$. También, definimos el **conjunto de** π como

$$\texttt{set}(\pi) = \{s_i \mid s_i \in \texttt{Markers}(\mathcal{X})\}$$

Propiedades ejec-SL. Tenemos que:

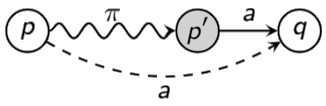
1. Para todo $i \neq j$, si $s_i = s_j$, entonces $s_i = s_j = \epsilon$.
2. Para todo par de ejec-SL distintas π_1, π_2 de p a q en $\mathcal{A}$, se cumple que $\texttt{set}(\pi_1) = \texttt{set}(\pi_2)$.

La demostraciones de estas propiedades queda como ejercicio propuesto para el lector.

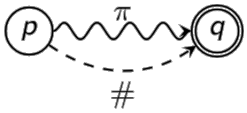
Demostración teorema 33. Dado un vset autómata funcional $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \Delta, I, F)$, construimos:

$$\mathcal{N} = \Big(Q, \Sigma \cup \{\#\}, 2^{\texttt{Markers}\ (\mathcal{X})}, \Delta', I, F\Big)$$

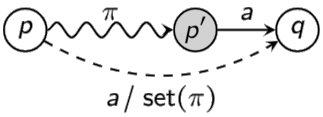
$(p, a, q) \in \Delta' \Leftrightarrow$ existe $p' \in Q$ y una ejec-SL π de p a p' tal que $(p', a, q) \in \Delta$ y $\texttt{set}(\pi) = \varnothing$.



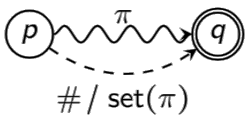
$(p, \#, q) \in \Delta' \Leftrightarrow$ existe una ejec-SL π de p a q tal que $\texttt{set}(\pi) = \varnothing$ y $q \in F$



$(p, a, S, q) \in \Delta' \Leftrightarrow$ existe $p' \in Q$ y ejec-SL π de p a p' tal que $(p', a, q) \in \Delta$ y $\texttt{set}(\pi) = S \neq \varnothing$.



$(p, \#, S, q) \in \Delta' \Leftrightarrow$ existe una ejec-SL π de p a q tal que $\texttt{set}(\pi) = S \neq \varnothing$ y $q \in F$

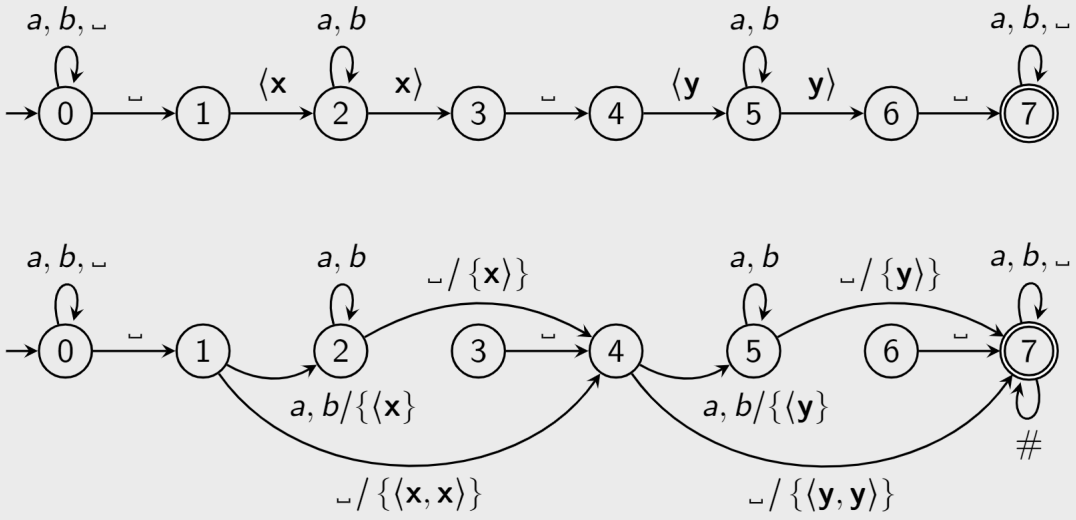


Por las propiedades 1 y 2, la construcción es **correcta**. Por último, podemos ver que

$$[\![\mathcal{N}]\!](d \cdot \#) = \{\texttt{seq}(\mu) \mid \mu \in [\![\mathcal{A}]\!](d)\}$$

■

Ejemplo 6.14: Construcción



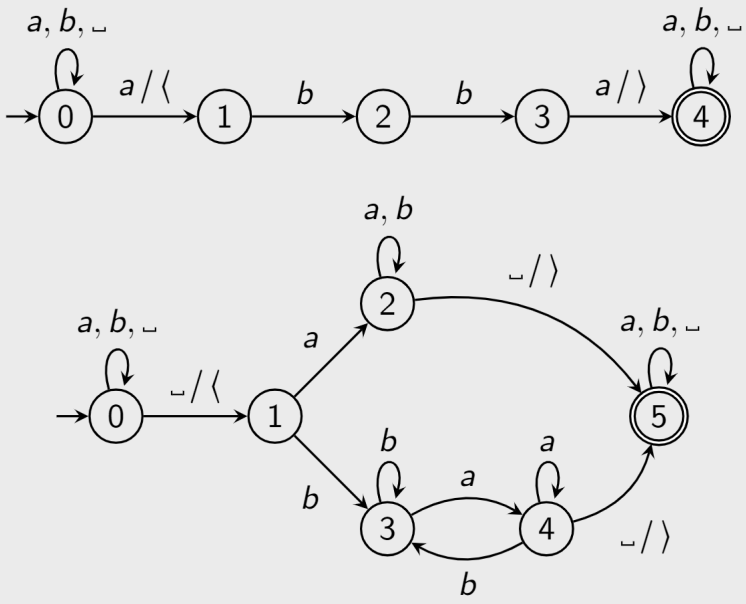
6.2.4. Determinismo

Sea $\mathcal{N} = (Q, \Sigma, \Lambda, \Delta, I, F)$ un autómata con anotaciones AnnA.

Definición. Decimos que $\mathcal{N}$ es **Input-Output** determinista (I/O-determinista) si, y sólo si, $|I| = 1$ y para todo $(p, t_1, q_1), (p, t_2, q_2) \in \Delta$, si $t_1 = t_2$, entonces $q_1 = q_2$.

$\mathcal{N}$ funciona de manera determinista al recibir el documento y una **anotación simultáneamente**.

Ejemplo 6.15



Teorema 34

Para todo AnnA $\mathcal{N} = (Q, \Sigma, \Lambda, \Delta, I, F)$, existe un AnnA I/O-determinista $\mathcal{N}^{det}$ tal que

$$[\![\mathcal{N}]\!] = [\![\mathcal{N}^{det}]\!]$$

Demostración teorema 34. Considere la determinización de $\mathcal{N}$ como:

$$\mathcal{A}^{\text{det}} = \left(2^Q, \Sigma, \Lambda, \Delta^{\text{det}}, q_0^{\text{det}}, F^{\text{det}}\right)$$

donde:

- $2^Q = \{S \mid S \subseteq Q\}$ es el conjunto potencia de Q.
- $q_0^{\text{det}} = I$
- $\Delta^{\text{det}} : 2^Q \times (\Sigma \cup \Sigma \times \Lambda) \to 2^Q$ tal que para todo $t \in \Sigma \cup (\Sigma \times \Lambda)$:

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

- $F^{\text{det}} = \left\{S \in 2^Q \mid S \cap F \neq \varnothing\right\}$. ■