Lenguajes Formales, Autómatas y Computabilidad

@valnrms

Apunte teórico de la materia Lenguajes Formales, Autómatas y Computabilidad de la Universidad Nacional de Buenos Aires, dictada en el segundo cuatrimestre de 2024.

Profesora: Verónica Becher. Gracias por el lindo cuatrimestre y las ganas puestas en dictar la materia.

Fecha de compilación: 16 / 02 / 2025.

Agradecimiento especial a Franco Frizzo, Leonardo Cremona, Elisa Orduna, Sabrina Silvero y Simón Lew Deveali por su aguante durante todo el cuatrimestre.
También a Pablo Barenbaum por ayudar a esclarecer conceptos que me resultaban confusos.

Este texto está principalmente hecho para mí.
De ninguna manera reemplaza cualquier material oficial que pueda tener la cátedra, ni mucho menos las clases teórias, ni prácticas, de la materia.
Se asume cierto entendimiento de las materias correlativas a esta.

Algo trivial, aunque es importante recalcar, este texto puede contener *orrorez*.

Índice

1. Lenguajes formales, gramáticas y la jerarquía de Chomsky

1.1. Alfabetos y palabras

Vamos a comenzar estudiando lenguajes. Pensemos, sin rompernos mucho la cabeza, en un lenguaje como el español. Este lenguaje está compuesto por palabras, que a su vez, las palabras, están compuestas por símbolos. Algunos ejemplos de símbolos son «!», «¿», «a», «A», «7», etc...

En la materia llamamos **alfabeto** Σ a un **conjunto finito**, **no vacío**, **de símbolos**.

Luego, las **palabras** se definen como la **secuencia finita** de **cero o más elementos** de un **alfabeto** Σ.
Por ejemplo, la palabra «hola» es una palabra sobre el alfabeto $\Sigma_{\text{hola}} = \{h, o, l, a\} = \{a, h, l, o\}$.

Uno ahora podría preguntarse, ¿y «alho»?, ¿es una palabra sobre el alfabeto Σ_{hola}? La respuesta es que sí. Actualmente estamos analizando la sintaxis, no la semántica de los lenguajes. «¿Y es necesario usar todos los elementos del alfabeto para formar una palabra?», no. Por ejemplo, «h» es una palabra sobre el alfabeto Σ_{hola}. Lo mismo ocurre para «hhhlaaaoaaa».

Si hilamos fino en la definición de palabra, se puede notar que dijimos **cero** o más elementos. Esto implica la existencia de la palabra vacía, que se denota como λ.

1.2. Clausura de Kleene sobre un alfabeto

Dado un alfabeto Σ, la **clausura de Kleene** del alfabeto Σ *(denotada como Σ^*)* es el conjunto de todas las palabras que se pueden formar con los elementos de Σ.
Es decir, donde $\Sigma^0 = \{\lambda\}$, $\Sigma^1 = \Sigma$, $\Sigma^2 = \Sigma \times \Sigma$, ... $\quad | \quad \Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup ... = \cup_{i \geq 0} \Sigma^i$.

Donde el producto de alfabetos, sigue la definición de producto cartesiano de cualquier conjunto.

1.2.1. Clausura positiva

La **clausura positiva** de un alfabeto Σ *(denotada como Σ^+)* es el conjunto de todas las palabras que se pueden formar con los elementos de Σ, excluyendo la palabra vacía.
Es decir, donde $\Sigma^1 = \Sigma$, $\Sigma^2 = \Sigma \times \Sigma$, ... $\quad | \quad \Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup ... = \cup_{i \geq 1} \Sigma^i$.

1.3. Cardinalidad de un alfabeto

«¿Cuál es la cardinalidad del conjunto Σ^i?»
El conjunto Σ tiene $|\Sigma|$ elementos. Para cada $i \geq 0$, Σ^i tiene $|\Sigma|^i$ elementos.

El conjunto Σ^* tiene una cantidad infinita numerable de palabras.
Es decir, $|\Sigma^*|$ es igual a la cardinalidad de $\mathbb{N}$. **← Teorema**

Demostración:
Pensemos en una relación de orden $\prec$, para $\Sigma^* \times \Sigma^*$.
Asumamos un orden lexicográfico entre los elementos del alfabeto.
Extendamos $\prec$ a un orden lexicográfico entre todas las palabras de la misma longitud (utilizando el orden $\prec$ del alfabeto), tal que las palabras de menor longitud son menores que las de mayor longitud.

Luego, definiamos la función biyectiva $f : \mathbb{N} \to \Sigma^*$ tal que $f(i) = i$-ésima palabra dada por el orden $\prec$. Por lo que podemos mapear cada palabra de Σ^* a un número natural. Y sabemos que el conjunto de los números naturales es infinito numerable.

□

1.4. Lenguaje sobre un alfabeto

Un **lenguaje** L sobre un alfabeto Σ es un **conjunto** de palabras que se pueden formar con los elementos de Σ. Es decir, $L \subseteq \Sigma^*$.
Ejemplos: $\emptyset$, $\{\lambda\}$ *(distinto a* $\emptyset$*)*, $\{0, 01, 011, 0111, 01111, ...\}$ es un lenguaje sobre $\Sigma = \{0, 1\}$.

1.4.1. ¿Cuántos lenguajes hay?

Para responder esta pregunta, va a ser importante recordar lo visto sobre la cardinalidad de un cojunto de partes.

1.4.1.1. Cardinalidad de un conjunto de partes

Dado un conjunto A, se define $\mathcal{P}(A)$ como **el conjunto de todos los subconjuntos** de A.
Sea A un conjunto finito, entonces $|\mathcal{P}(A)| = 2^{|A|}$.

Luego, si Σ es un alfabeto, $|\mathcal{P}(\Sigma^*)| = 2^{\mathbb{N}}$.

1.4.1.2. La cantidad de lenguajes es no numerable

$|\Sigma^*| < |\mathcal{P}(\Sigma^*)|$ **← Teorema**

Demostración:
Sale usando el argumento de la diagonal de Cantor.
Supongamos que $|\mathcal{P}(\Sigma^*)|$ es infinito numerable. Entonces, podemos enumerar los lenguajes como $L_1, L_2, L_3, ...$
Y demostramos anteriormente que $|\Sigma^*|$ es infinito numerable. Por lo que cualquier subconjunto *(lenguaje)* de Σ^* es numerable.
Por lo que para cada $L_1, L_2, L_3, ...$ podemos numerar sus palabras. Luego,

$$L_1 : w_{1,1}, w_{1,2}, w_{1,3}, ...$$
$$L_2 : w_{2,1}, w_{2,2}, w_{2,3}, ...$$
$$L_3 : w_{3,1}, w_{3,2}, w_{3,3}, ...$$
$$...$$

Definamos $\mathcal{L} = \{u_1, u_2, u_3, ...\}$ donde $u_1 \prec u_2 \prec u_3 \prec ...$, y $\forall i, w_{i,i} \prec u_i$.
Notar que $\mathcal{L}$ debe estar en nuestra tablita de arriba, pues es un lenguaje y la tablita que creamos suponiendo que $|\mathcal{P}(\Sigma^*)|$ es numerable, contiene todos los lenguajes.
Luego, la i-ésima palabra de $\mathcal{L}$ difiere de la i-ésima palabra de L_i en la i-ésima posición, para cualquier i.
Por lo que $\mathcal{L}$ no es ninguno de los L_i (o sea, no puede estar en nuestra tablita), lo que es una contradicción. **¡ABSURDO!**
¿De dónde vino el absurdo?, de suponer que $|\mathcal{P}(\Sigma^*)|$ era infinito numerable.

□

1.5. Gramáticas

Una gramática es una 4-upla $G = \langle V_N, V_T, P, S \rangle$ donde

- V_N es un conjunto de **símbolos no terminales**.
- V_T es un conjunto de **símbolos terminales**.
- P es un conjunto de **producciones**, que es un conjunto finito de

$$(V_N \cup V_T)^* V_N (V_N \cup V_T)^* \times (V_N \cup V_T)^*,$$

estas producciones son entonces pares ordenados (α, β), que usualmente son notados como $\alpha \to \beta$.

- $S \in V_N$ es el **símbolo inicial**.

Por ejemplo, sea $G = \langle V_N, V_T, P, S \rangle$ la gramática *(en particular, libre de contexto; se profundiza después)* tal que

- $V_N = \{S\}$
- $V_T = \{+, *, a, (,)\}$
- P:
  - $S \to S + S$
  - $S \to S * S$
  - $S \to (S)$
  - $S \to a$

1.5.1. Lenguaje generado por una gramática

Generalmente solemos utilizar la siguiente frase: **«Las gramáticas generan *(lenguajes)*, los autómatas reconocen *(lenguajes)* y las expresiones regulares describen *(lenguajes)***».

Es por ello que una gramática, como la que definimos anteriormente, genera un lenguaje. Al lenguaje generado por una gramática G lo denotamos como $\mathcal{L}(G)$.

Dada una gramática $G = \langle V_N, V_T, P, S \rangle$, se define el lenguaje generado por G como

$$\mathcal{L}(G) = \left\{ w \in V_T^* : S \overset{+}{\underset{G}{\Rightarrow}} w \right\}$$

donde $\overset{+}{\underset{G}{\Rightarrow}}$ es derivación en uno o más pasos, que se obtiene de la clausura transitiva de la derivación directa $\underset{G}{\Rightarrow}$.

1.5.2. Derivación directa

Si $\alpha\beta\gamma \in (V_N \cup V_T)^*$ y $(\beta \to \delta) \in P$, entonces $\alpha\beta\gamma$ se deriva directamente en $\alpha\delta\gamma$.
Esto lo denotamos como:

$$\alpha\beta\gamma \underset{G}{\Rightarrow} \alpha\delta\gamma$$

Entonces, $\underset{G}{\Rightarrow}$ es una relación sobre $(V_N \cup V_T)^*$, es decir, $\underset{G}{\Rightarrow} \subseteq (V_N \cup V_T)^* \times (V_N \cup V_T)^*$.
Podemos componer la relación $\underset{G}{\Rightarrow}$ consigo misma, cero o más veces; a su vez, podemos componerla para obtener la clausura transitiva de la derivación directa, denotada como $\overset{+}{\underset{G}{\Rightarrow}}$.

1.5.2.1. Clausura de Kleene de la relación de derivación

La relación $\mathrm{id}_{(V_N \cup V_T)^*}$ se define como:

$$\alpha \, \mathrm{id}_{(V_N \cup V_T)^*} \, \beta \iff \alpha = \beta.$$

donde $\alpha, \beta \in (V_N \cup V_T)^*$, y vale para cualquier α, β.

La clausura de Kleene de la relación de derivación $\underset{G}{\Rightarrow}$ se define como:

$$\left(\underset{G}{\Rightarrow}\right)^0 = \mathrm{id}_{(V_N \cup V_T)^*}$$

Luego,

$$\left(\underset{G}{\Rightarrow}\right)^* = \left(\underset{G}{\Rightarrow}\right)^+ \cup \mathrm{id}_{(V_N \cup V_T)^*} = \cup_{i \geq 0} \left(\underset{G}{\Rightarrow}\right)^i$$

1.5.3. Forma sentencial de una gramática

Una forma sentencial de una gramática es **cualquier cadena de símbolos** que se puede generar a partir del **símbolo inicial** (S) de la gramática mediante una serie de derivaciones aplicando las reglas de producción de la gramática.

Es decir, sea $G = \langle V_N, V_T, P, S \rangle$ una gramática:

- S es una **forma sentencial** de G.
- Si $\alpha\beta\gamma$ es una forma sentencial de G, y $(\beta \to \delta) \in P$, entonces $\alpha\delta\gamma$ es también una forma sentencial de G.

Las formas sentenciales están formadas por elementos de $(V_N \cup V_T)^*$.

1.5.4. Derivaciones a izquierda y a derecha

Al realizar una derivación, podemos elegir entre derivar a izquierda o a derecha.

- Derivación más a la izquierda $\underset{L}{\Rightarrow}$.
- Derivación más a la derecha $\underset{R}{\Rightarrow}$.

Para hacer i pasos de una derivación escribimos, respectivamente, $\overset{i}{\underset{L}{\Rightarrow}}$ y $\overset{i}{\underset{R}{\Rightarrow}}$.
Para la clausura transitiva de la derivación, escribimos $\overset{+}{\underset{L}{\Rightarrow}}$ y $\overset{+}{\underset{R}{\Rightarrow}}$.
Y para la clausura de Kleene de la derivación, escribimos $\overset{*}{\underset{L}{\Rightarrow}}$ y $\overset{*}{\underset{R}{\Rightarrow}}$.

1.5.5. Árboles de derivación

Asumamos una gramática $G = \langle V_N, V_T, P, S \rangle$.

1.5.5.1. Definición

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

Las etiquetas de las hojas están en $V_T \cup \{\lambda\}$, y las etiquetas de los nodos internos están en V_N.
Las etiquetas de sus hijos son los símbolos del cuerpo de una producción (lado derecho).

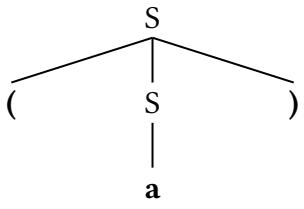
Un nodo tiene etiqueta A y tiene n descendientes etiquetados $X_1, X_2, ..., X_n$, exactamente cuando hay una derivación que usa una producción $A \to X_1 X_2 ... X_n$.

Si no se entiende mucho, no te preocupes. Lo vas a entender mejor con un ejemplo.

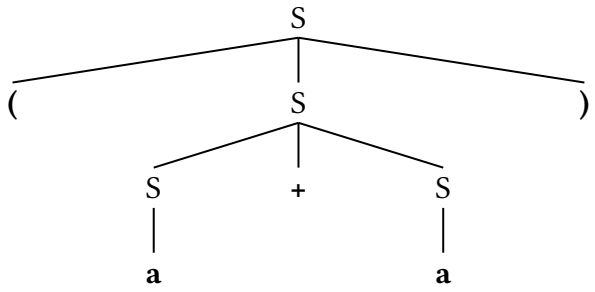
1.5.5.2. Ejemplo

Sea $G = \langle \{S\}, \{+, *, a, (,)\}, P, S\rangle$ la gramática libre de contexto con P:

- $S \to S + S$
- $S \to S * S$
- $S \to (S)$
- $S \to a$



$S \underset{L}{\Rightarrow} (S) \underset{L}{\Rightarrow} (a)$



$S \underset{L}{\Rightarrow} (S) \underset{L}{\Rightarrow} (S+S) \underset{L}{\Rightarrow} (a+S) \underset{L}{\Rightarrow} (a+S) \underset{L}{\Rightarrow} (a+a)$

1.5.5.3. Camino de un árbol de derivación

Dada una gramática $\langle V_N, V_T, P, S\rangle$ y dados $X \in V_T$ o $X \in V_N$.
Llamamos **camino de X en un árbol $\mathcal{T}(A)$** con $A \in V_N$, a la secuencia $A, X_1, ..., X_k, X$ que corresponde a las etiquetas de los vértices de la rama del árbol.
Una hoja de un árbol $\mathcal{T}(A)$, es un símbolo $t \in V_T$ para el cual hay un camino que empieza en A y termina en t.

1.5.5.4. Altura de un árbol de derivación

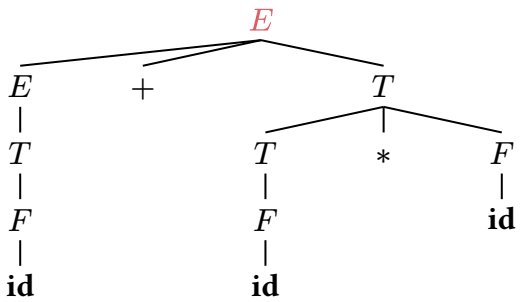
La altura de $\mathcal{T}(A)$, con $A \in V_N$, es

$$\text{máx}\{|\alpha x| : A\alpha x \text{ es un camino de } x \wedge x \text{ es una hoja de } \mathcal{T}(A)\}.$$

Consideremos este árbol de derivación para $\text{id} + \text{id} * \text{id}$ en la gramática libre de contexto $G = \langle V_N, V_T, P, E\rangle$ con $V_N = \{E, T, F\}$ y $V_T = \{\mathbf{id}, \mathbf{const}, +, *, (,)\}$, y las producciones:

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to \mathbf{id} \mid \mathbf{const} \mid (E)$$

Este es un árbol de derivación para $\text{id} + \text{id} * \text{id}$ en la gramática G (y es único):



La altura de $\mathcal{T}(A) = \text{máx}\{|\alpha x| : A\alpha x$ es un camino de $x \wedge x$ es una hoja de $\mathcal{T}(A)\}$.
Luego, $A = E$, $\alpha = E, T, F$ y $x = \mathbf{id}$, por lo tanto, la altura es 4.

1.5.5.5. Lema sobre la cota de la longitud de una palabra a partir de la altura de un árbol de derivación

Sea $G = \langle V_N, V_T, P, S\rangle$ una gramática libre de contexto con $P \neq \emptyset$, sea $\alpha \in (V_N \cup V_T)^*$ y sea $\mathcal{T}(S)$ un árbol de derivación para α en G cuya altura designaremos h.

Si

$$a = \text{máx}\{k : (k = |\beta|, (A \to \beta) \in P, \beta \neq \lambda) \vee (k = 1, (A \to \lambda) \in P)\},$$

en término de árbol, a es la cantidad máxima de hijos que puede tener un nodo

entonces $a^h \geq |\alpha|$.

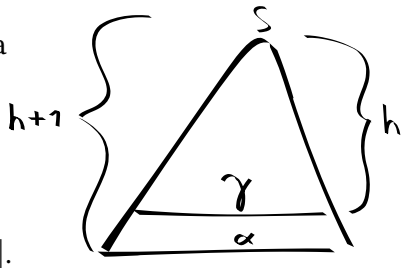
1.5.5.6. Demostración del lema

Por inducción en h.

- **Caso base:** $h = 0$.
  El único árbol de derivación posible es el símbolo distinguido S, cuya altura es 0.
  Por lo tanto, $a^h = a^0 = 1 \geq 1 = |S|$.
- **Paso inductivo:** $h > 0$.
  Asumimos que vale para h, probemos que vale para $h + 1$.
  Sea γ la base del árbol $\mathcal{T}(S)$ para la altura h. Asumamos la **H.I.**: $a^h \geq |\gamma|$.

  Sea α la base de $\mathcal{T}(S)$ para la altura $h + 1$.

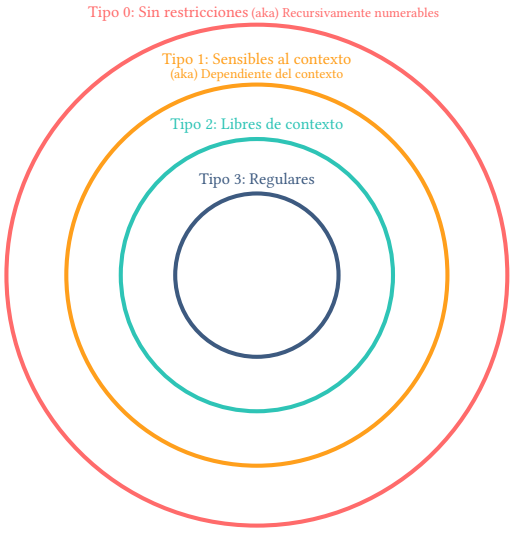
  Luego, $a|\gamma| \geq |\alpha|$.
  Pero, por **H.I.**, $a^h \geq |\gamma|$, entonces $a^{h+1} = aa^h \geq a|\gamma| \geq |\alpha|$.



□

1.6. Jerarquía de Chomsky

La **jerarquía de Chomsky** es una clasificación de las gramáticas formales en cuatro clases, según su capacidad de generar lenguajes formales.



1.6.1. Gramáticas de tipo 0 (gramáticas sin restricciones)

$$\alpha \to \beta, \text{ con } \alpha, \beta \in (V_N \cup V_T)^*$$

1.6.2. Gramáticas de tipo 1 (gramáticas sensibles al contexto)

$$\alpha \to \beta, \text{ con } \alpha, \beta \in (V_N \cup V_T)^* \text{ y } |\alpha| \leq |\beta|$$

1.6.3. Gramáticas de tipo 2 (gramáticas libres de contexto)

$$A \to \beta, \text{ con } A \in V_N \text{ y } \beta \in (V_N \cup V_T)^*$$

1.6.4. Gramáticas de tipo 3 (gramáticas regulares)

$$A \to a,\ A \to aB,\ A \to \lambda, \text{ con } A, B \in V_N \text{ y } a \in V_T$$

La jerarquía de gramáticas da origen a la jerarquía de los lenguajes.

1.7. Algunos lemas sobre gramáticas

Definición de una gramática no recursiva a izquierda:

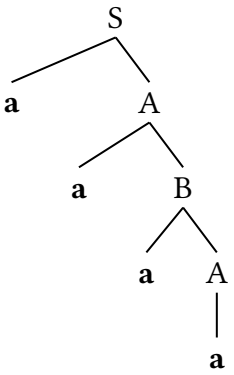
Una gramática no es recursiva a izquierda si no tiene derivaciones $A \overset{+}{\underset{L}{\Rightarrow}} A\alpha$, para ningún $\alpha \in (V_N \cup V_T)^*$.

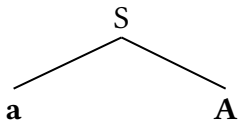
1.7.1. Regular

Sea $G = \langle V_N, V_T, P, S\rangle$ regular, no recursiva a izquierda.
Si $A \overset{i}{\underset{L}{\Rightarrow}} wB$, entonces $i = |w|$.

Demostración:
Consideremos el siguiente árbol de derivación para $w = a_1 a_2 ... a_n$:

Si agarramos el árbol de derivación que vemos a la derecha, y realizamos un corte en la altura i, obtenemos un subarbol con i hojas, $a_1 ... a_i$, donde el único nodo que no es hoja tiene como etiqueta un símbolo $\in V_N$.





← Ejemplo con $i = 1$.

A su vez, debido a la forma de las producciones, cada derivación agrega uno, y solo uno, símbolo terminal.
Por lo que en i derivaciones, se agregan los i símbolos terminales y el no terminal de la derecha.

□

P:

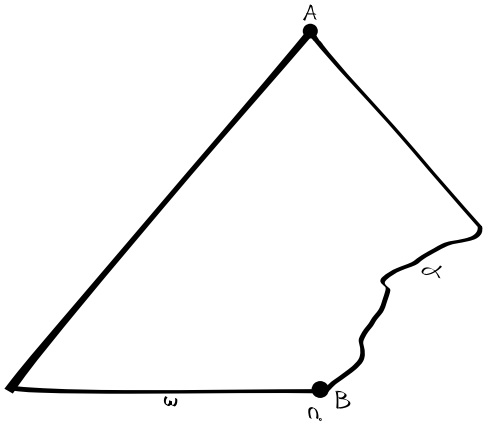
$$S \to aA \mid \lambda$$
$$A \to a \mid aB$$
$$B \to aA$$

1.7.2. Libre de contexto

Sea $G = \langle V_N, V_T, P, S\rangle$ una gramática libre de contexto, no recursiva a izquierda.
Existe una constante c tal que si $A \overset{i}{\underset{L}{\Rightarrow}} wB\alpha$ entonces $i \leq c^{|w|+2}$,
donde $A, B \in V_N$, $w \in V_T^*$ y $\alpha \in (V_N \cup V_T)^*$.

Demostración:

Llamemos $k = |V_N|$ y $n = |w|$. Consideremos el árbol de derivación más a la izquierda para $A \overset{i}{\underset{L}{\Rightarrow}} wB\alpha$.



Sea n_0 el nodo con etiqueta B en la derivación $A \underset{L}{\overset{i}{\Rightarrow}} wB\alpha$.
Por ser la derivación más a la izquierda, todos los caminos a la derecha del camino desde la raíz a n_0 son más cortos, o del mismo largo.
Supongamos que hay un camino de longitud mayor o igual que $(n+2)k$ nodos de la raíz a la hoja, es decir:
Sea $A = X_0 \frown X_1 \frown ... \frown X_{(n+2)k} = B$, el camino desde la raíz a n_0.
Separamoslos en $n+2$ segmentos de $k+1$ nodos,

$$A = X_0 \frown ... \frown X_k, \quad X_k \frown ... \frown X_{2k}, \quad ..., \quad X_{(n+1)k} \frown ... \frown X_{(n+2)k} = B.$$

Notar que se repite el final de uno con el principio del siguiente.

Consideremos los $n+2$ subárboles de derivación, con $\beta \in (V_N \cup V_T)^*$ apropiados.

$$A = X_0 \underset{L}{\overset{k}{\Rightarrow}} \beta_1 X_k, \quad X_k \underset{L}{\overset{k}{\Rightarrow}} \beta_2 X_{2k}, \quad ..., \quad X_{(n+1)k} \underset{L}{\overset{k}{\Rightarrow}} \beta_{n+2} X_{(n+2)k}.$$

Es imposible que cada uno produzca uno o más símbolos de wB, porque $|wB| = n+1$.
Al menos uno no produce ningún símbolo de wB,

$$X_{jk} \underset{L}{\Rightarrow} \beta_{jk+1} X_{jk+1} \underset{L}{\Rightarrow} ... \underset{L}{\Rightarrow} \beta_{(j+1)k} X_{(j+1)k}$$

y deriva solamente λ.

Cada uno de $X_{jk}, \beta_{jk+1} X_{jk+1}, ..., \beta_{(j+1)k} X_{(j+1)k}$ empiezan con un símbolo de V_N, son en total $k+1$.
Necesariamente hay dos que empiezan con el mismo símbolo. Pero esto contradice que la gramática no es recursiva a izquierda.
Entonces, nuestra suposición de que el camino de la raíz a n_0 tiene $n+2$ segmentos de $k+1$ nodos es imposible.

Concluimos que su longitud es menor que $(n+2)k$.
Sea ℓ el máximo número de símbolos en la parte derecha de una producción de la gramática, la cantidad de nodos del árbol de derivación es a lo sumo $\ell^{k(n+2)}$.

Por lo tanto, $A \underset{L}{\overset{i}{\Rightarrow}} wB\alpha$ con $i \leq \ell^{k(n+2)}$.

Para finalizar la demostración basta tomar $c = \ell^k$.

□

2. Autómatas finitos determinísticos, no determinísticos y gramáticas regulares

2.1. Autómata finito determinístico (AFD)

Un **autómata finito determinístico (AFD)** es una máquina abstracta definida formalmente como una 5-upla $\langle Q, \Sigma, \delta, q_0, F\rangle$ donde

- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada.
- $\delta : Q \times \Sigma \to Q$ es la función de transición.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

2.1.1. Función de transición generalizada $\hat{\delta}$

Definimos $\hat{\delta} : Q \times \Sigma^* \to Q$,

- $\hat{\delta}(q, \lambda) = q$, para todo $q \in Q$.
- $\hat{\delta}(q, xa) = \delta\Big(\hat{\delta}(q, x), a\Big)$, con $q \in Q$, $x \in \Sigma^*$ y $a \in \Sigma$.

Notar que $\hat{\delta}(q, a) = \delta\Big(\hat{\delta}(q, \lambda), a\Big) = \delta(q, a)$.
Muchas veces usamos el símbolo δ para ambas funciones. **← Abuso de notación**

2.1.2. Lenguaje aceptado por un AFD

El lenguaje aceptado por un AFD $M = \langle Q, \Sigma, \delta, q_0, F\rangle$, al que denotamos como $\mathcal{L}(M)$, es el conjunto de cadenas de Σ^* aceptadas por M.

$$\mathcal{L}(M) = \Big\{x \in \Sigma^* : \hat{\delta}(q_0, x) \in F\Big\}.$$

Entendemos a los autómatas finitos como funciones tales que para cada cadena dan un valor booleano: la aceptación o la no aceptación,

$$M : \Sigma^* \to \{0, 1\}.$$

2.1.3. Configuración instantánea de un AFD

Sea AFD $M = \langle Q, \Sigma, \delta, q_0, F\rangle$. Una configuración instantánea de M es un par (q, x) en $Q \times \Sigma^*$, donde $q \in Q$ es el estado actual y $x \in \Sigma^*$ es la cadena de entrada restante por procesar.

2.1.4. Transición entre configuraciones instantáneas $\vdash$

Llamamos transición a la siguiente relación sobre $Q \times \Sigma^*$:

$$(q, x) \vdash (p, y) \text{ si } (\delta(q, a) = p \land x = ay).$$

De lo anterior tenemos que $(q, xy) \overset{*}{\vdash} (p, y)$ si y solo si $\hat{\delta}(q, x) = p$, es decir, si se puede pasar del estado q al estado p leyendo la cadena x.

2.2. Autómata finito no determinístico (AFND)

Un **autómata finito no determinístico (AFND)** es una máquina abstracta definida formalmente como una 5-upla $\langle Q, \Sigma, \delta, q_0, F\rangle$ donde

- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada.

- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ es la función de transición.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

2.2.1. Función de transición generalizada $\hat{\delta}$

Definimos $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$,
- $\hat{\delta}(q, \lambda) = \{q\}$, para todo $q \in Q$.
- $\hat{\delta}(q, xa) = \Big\{p \in Q : \exists r \in \hat{\delta}(q, x) \land p \in \delta(r, a)\Big\}$ con $q \in Q$, $x \in \Sigma^*$ y $a \in \Sigma$.

Notar que

$$\begin{aligned}\hat{\delta}(q, \lambda a) &= \Big\{p \in Q : \exists r \in \hat{\delta}(q, \lambda) \land p \in \delta(r, a)\Big\} \\ &= \{p \in Q : \exists r \in \{q\} \land p \in \delta(r, a)\} \\ &= \{p \in Q : p \in \delta(q, a)\} \\ &= \delta(q, a).\end{aligned}$$

Nuevamente, **abuso de notación**, muchas veces usamos el símbolo δ para ambas funciones.

2.2.2. Lenguaje aceptado por un AFND

El lenguaje aceptado por un AFND $M = \langle Q, \Sigma, \delta, q_0, F\rangle$, al que denotamos como $\mathcal{L}(M)$, es el conjunto de cadenas de Σ^* aceptadas por M.

$$\mathcal{L}(M) = \Big\{x \in \Sigma^* : \hat{\delta}(q_0, x) \cap F \neq \emptyset\Big\}.$$

2.2.3. Configuración instantánea de un AFND

Sea AFND $M = \langle Q, \Sigma, \delta, q_0, F\rangle$. Una configuración instantánea de M es un par (q, x) en $Q \times \Sigma^*$, donde $q \in Q$ es el estado actual y $x \in \Sigma^*$ es la cadena de entrada restante por procesar.

2.2.4. Transición entre configuraciones instantáneas $\vdash$

Llamamos transición a la siguiente relación sobre $Q \times \Sigma^*$:

$$(q, x) \vdash (p, y) \text{ si } (p \in \delta(q, a) \land x = ay).$$

2.2.5. Función de transición de conjuntos de estados en un AFND

Podemos extender la función de transición aún más, haciendo que mapee conjuntos de estados y cadenas en conjuntos de estados.

- $\delta : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$ dada por $\delta(P, a) = \bigcup_{q \in P} \delta(q, a)$.
- $\hat{\delta} : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ dada por $\hat{\delta}(P, x) = \bigcup_{q \in P} \hat{\delta}(q, x)$.

La primer extensión nos permite escribir $\hat{\delta}(q, xa)$ como

$$\hat{\delta}(q, xa) = \delta\Big(\hat{\delta}(q, x), a\Big).$$

Si estás confuso al leer esto no te preocupes.
Es muy común tratar ambas extensiones como δ indistintamente, diferenciándolas por el contexto. Si es $Q \times \Sigma$, $\mathcal{P}(Q) \times \Sigma$, $\mathcal{P}(Q) \times \Sigma^*$ o $Q \times \Sigma^*$.
Generalmente en las demostraciones vamos a usar δ sin importar si es:
- $\delta : Q \times \Sigma \to Q$,

- $\delta : Q \times \Sigma^* \to Q$,
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$.
- $\delta : Q \times \Sigma^* \to \mathcal{P}(Q)$.
- $\delta : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$,
- $\delta : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$.

Y lo diferenciamos por el contexto.

En la siguiente sección (AFND-λ) vamos a ver más extensiones sobre δ. De igual forma, vamos a seguir abusando de la notación.

2.3. Autómata finito no determinístico con transiciones λ (AFND-λ)

Un **autómata finito no determinístico con transiciones lambda (AFND-λ)** es un AFND que puede hacer transiciones sin consumir ningún símbolo de entrada, es definido formalmente como una 5-upla $\langle Q, \Sigma, \delta, q_0, F\rangle$ donde

- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada, tal que $\lambda \notin \Sigma$.
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q)$ es la función de transición.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

2.3.1. Clausura λ

La clausura λ de un estado q, $Cl_\lambda(q)$, es el conjunto de estados alcanzables desde q, siguiendo sólo transiciones λ.
Usamos la noción de clausura transitivo-reflexiva para definir Cl_λ.

2.3.2. Clausura λ de un estado

Dado un AFND-λ $\langle Q, \Sigma, \delta, q_0, F\rangle$, y sea $R \subseteq Q \times Q$ tal que $(q, p) \in R$ si y solo si $p \in \delta(q, \lambda)$. Definimos $Cl_\lambda : Q \to \mathcal{P}(Q)$ como

$$Cl_\lambda(q) = \{p \in Q : (q, p) \in R\}.$$

Notar que $q \in Cl_\lambda(q)$.

2.3.3. Clausura λ de un conjunto de estados P

$$Cl_\lambda(P) = \bigcup_{q \in P} Cl_\lambda(q).$$

Notar que una es $Cl_\lambda : \mathcal{P}(Q) \to \mathcal{P}(Q)$ y la otra es $Cl_\lambda : Q \to \mathcal{P}(Q)$.

Extendemos la definición de δ a un conjunto de estados para AFND-λ,

$$\delta : \mathcal{P}(Q) \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q) \text{ dada por } \delta(P, a) = \bigcup_{q \in P} \delta(q, a).$$

2.3.4. Función de transición $\hat{\delta}$ (sin λ)

Dado un AFND $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ con $\delta : Q \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q)$, definimos la función de transición sin λ, $\hat{\delta} : Q \times \Sigma \to \mathcal{P}(Q)$, como

$$\hat{\delta}(q, a) = Cl_\lambda(\delta(Cl_\lambda(q), a)).$$

Notar que $\hat{\delta}(q, a)$ puede ser distinto de $\delta(q, a)$.

Extendemos $\hat{\delta}$ a conjuntos de estados,

$$\hat{\delta} : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q) \text{ dada por } \hat{\delta}(P, a) = \bigcup_{q \in P} \hat{\delta}(q, a).$$

Extendemos $\hat{\delta}$ a palabras, $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$:

$$\hat{\delta}(q, \lambda) = Cl_\lambda(q),$$
$$\hat{\delta}(q, xa) = \hat{\delta}\Big(\hat{\delta}(q, x), a\Big).$$

2.3.5. Lenguaje aceptado por un AFND-λ

Sea AFND-λ $M = \langle Q, \Sigma, \delta, q_0, F\rangle$. El lenguaje aceptado por M, $\mathcal{L}(M)$, es el conjunto de cadenas aceptadas por M,

$$\mathcal{L}(M) = \Big\{x \in \Sigma^* : \hat{\delta}(q_0, x) \cap F \neq \emptyset\Big\}.$$

2.3.6. Equivalencia entre AFND y AFND-λ

Dado un AFND-λ $M = \langle Q, \Sigma, \delta, q_0, F\rangle$, podemos construir un AFND equivalente $M' = \langle Q', \Sigma, \delta', q_{0'}, F'\rangle$ tal que $\mathcal{L}(M) = \mathcal{L}(M')$.

Demostración:

Sea AFND-λ $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ donde $\delta : Q \times (\Sigma \cup \{\delta\}) \to \mathcal{P}(Q)$.

Definimos AFND $M' = \langle Q, \Sigma, \delta', q_0, F'\rangle$ donde $\delta' : Q \times \Sigma \to \mathcal{P}(Q)$,

Vamos a usar las versiones extendidas de δ y $\hat{\delta}$, la de conjunto de estados y palabras de Σ^*.

$$\delta'(q, a) = \hat{\delta}(q, a), \text{ para cada } a \in \Sigma \text{ y } q \in Q,$$
$$F' = \begin{cases} F & \text{si } Cl_\lambda(q_0) \cap F = \emptyset \\ F \cup \{q_0\} & \text{si no.} \end{cases}.$$

Observar que $F' \supseteq F$.

Debemos ver que $\forall x \in \Sigma^*, x \in \mathcal{L}(M) \iff x \in \mathcal{L}(M')$.

Demostración:

Caso $|x| = 0$: i.e., $x = \lambda$.

$\Rightarrow$) Supongamos que $x = \lambda \in \mathcal{L}(M)$. Eso quiere decir que: o $q_0 \in F$, o que existe alguna transición λ desde q_0 a algún estado final.
Más formalmente,

$$\begin{aligned} \lambda \in \mathcal{L}(M) &\Rightarrow \hat{\delta}(q_0, \lambda) \cap F \neq \emptyset \\ &\Rightarrow Cl_\lambda(q_0) \cap F \neq \emptyset \\ &\Rightarrow F' = F \cup \{q_0\} \\ &\Rightarrow q_0 \in F' \\ &\Rightarrow \delta'(q_0, \lambda) = \{q_0\} \cap F' \neq \emptyset \\ &\Rightarrow \lambda \in \mathcal{L}(M'). \end{aligned}$$

$\Leftarrow$) Supongamos que $x = \lambda \in \mathcal{L}(M')$. Eso quiere decir que $q_0 \in F'$, por lo que o $Cl_\lambda(q_0) \cap F \neq \emptyset$ o $q_0 \in F$.
Más formalmente,

$$\begin{aligned}\lambda \in \mathcal{L}(M') &\Rightarrow \delta'(q_0, \lambda) = \{q_0\} \cap F' \neq \emptyset \\ &\Rightarrow q_0 \in F' \\ &\Rightarrow F' = F \cup \{q_0\} \vee q_0 \in F \\ &\Rightarrow Cl_\lambda(q_0) \cap F \neq \emptyset \vee q_0 \in F \\ &\Rightarrow \hat{\delta}(q_0, \lambda) \cap F \neq \emptyset \vee q_0 \in F \\ &\Rightarrow \lambda \in \mathcal{L}(M).\end{aligned}$$

Caso $|x| \geq 1$:

Queremos ver que $x \in \mathcal{L}(M) \iff x \in \mathcal{L}(M')$. Demostremos antes que $\delta'(q, x) = \hat{\delta}(q, x)$, $\forall q \in Q$ y $\forall x \in \Sigma^+$.

Hagamos inducción en el largo de la cadena x.

Caso base: $|x| = 1$, i.e., $x = a$ donde $a \in \Sigma$.
Nos queda

$$\delta'(q, a) \underset{\text{definición de } \delta' \text{ en } M'}{=} \hat{\delta}(q, a).$$

Caso inductivo: $|x| = n + 1$, i.e., $x = ya$ donde $y \in \Sigma^*$ y $a \in \Sigma$.

$$\begin{aligned}\delta'(q, ya) &= \delta'(\delta'(q, y), a) \\ &\underset{\text{HI}}{=} \delta'\left(\hat{\delta}(q, y), a\right) \\ &= \bigcup_{p \in \hat{\delta}(q,y)} \delta'(p, a) \\ &= \bigcup_{p \in \hat{\delta}(q,y)} \hat{\delta}(p, a) \\ &= \hat{\delta}\left(\hat{\delta}(q, y), a\right) \\ &= \hat{\delta}(q, ya).\end{aligned}$$

□

2.4. Equivalencia entre AFD y AFND

Es trivial ver que, **para todo AFD podemos construir un AFND equivalente**; solamente modificamos la función de transición, de manera tal que para cada transición $\delta(q, a) = p$ en el AFD, tenemos $\delta(q, a) = \{p\}$ en el AFND.

Lo que no es tan obvio es que lo recíproco también es cierto: **para cada AFND existe un AFD equivalente**.

Dado un AFND $M = \langle Q, \Sigma, \delta, q_0, F\rangle$, podemos construir un AFD $M' = \langle Q', \Sigma, \delta', q_{0'}, F'\rangle$ tal que $\mathcal{L}(M) = \mathcal{L}(M')$. **← Teorema**

Demostración:
Sea $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ un AFND, definamos el AFD $M' = \langle Q', \Sigma, \delta', q_0', F'\rangle$ veamos cómo se construye.

- Los estados del autómata M', vamos a denotarlos como $[q_1, ..., q_i]$, con $q_1, ..., q_i \in Q$. Notar que son subconjuntos de Q, es decir $Q' = \mathcal{P}(Q)$.
- El Σ del autómata M' es el mismo que el de M, se ve en la definición de la 5-upla.

- Para las transiciones de M', vamos a definir la función de transición $\delta' : Q \times \Sigma \to Q$ como $\delta'([q_1, ..., q_i], a) = [p_1, ..., p_j]$ si y solo si $\delta(\{q_1, ..., q_i\}, a) = \{p_1, ..., p_j\}$.
- El estado inicial de M' lo vamos a definir como $q'_0 = [q_0]$
- Para los estados finales de M', vamos a tomar aquellos nodos $[q_1, ..., q_i]$ tales que alguno de los elementos en $\{q_1, ..., q_i\}$, es estado final en M.
  Luego, $F' = \{[q_1, ..., q_i] \in Q' : \{q_1, ..., q_i\} \cap F \neq \emptyset\}$.

Bueno, con esto ya tenemos definido el AFD M'. Falta probar que $\mathcal{L}(M) = \mathcal{L}(M')$.

Para ello vamos a necesitar **un lema**, probemos que

$$\delta'(q'_0, x) = \delta'([q_0], x) = [r_1, ..., r_k] \iff \delta(\{q_0\}, x) = \delta(q_0, x) = \{r_1, ..., r_k\},$$

simplificando,

$$\delta'(q'_0, x) = [r_1, ..., r_k] \iff \delta(q_0, x) = \{r_1, ..., r_k\}.$$

Asumiendo que $\delta'([q_1, ..., q_i], a) = [p_1, ..., p_j] \underset{\text{definición de } \delta'}{\iff} \delta(\{q_1, ..., q_j\}, a) = \{p_1, ..., p_i\}$ es verdadero.

Vamos a hacerlo por inducción en la longitud de la cadena x.

Caso base: $|x| = 0$, i.e., $x = \lambda$.

$$\delta'(q'_0, \lambda) = \delta'([q_0], \lambda) = [q_0] \iff \{q_0\} = \delta(q_0, \lambda) = \delta(\{q_0\}, \lambda).$$

El caso base es verdadero.

Paso inductivo: asumimos que vale para x tal que $|x| = n$, probemos que vale para $|x| = n + 1$.

Queremos probar que

$$\delta'(q'_0, xa) = [s_1, ..., s_l] \iff \delta(q_0, xa) = \{s_1, ..., s_l\}.$$

Asumiendo que $\delta'(q'_0, x) = [r_1, ..., r_k] \underset{\text{HI}}{\iff} \hat{\delta}(q_0, x) = \{r_1, ..., r_k\}$ es verdadero.

Entonces,

$$\delta'(q'_0, xa) \underset{\text{por definición de } \delta' \text{ en } M'}{=} \delta'(\delta'(q'_0, x), a) = [s_1, ..., s_l]$$

$$\iff$$

$$\exists [r_1, ..., r_k] : \delta'(q'_0, x) = [r_1, ..., r_k] \land \delta'([r_1, ..., r_k], a) = [s_1, ..., s_l]$$

$$\underset{\text{por HI y def. de } \delta' \text{ en } M'}{\iff}$$

$$\exists [r_1, ..., r_k] : \delta(q_0, x) = \{r_1, ..., r_k\} \land \delta(\{r_1, ..., r_k\}, a) = \{s_1, ..., s_l\}$$

$$\iff$$

$$\delta(q_0, xa) = \delta(\delta(q_0, x), a) = \{s_1, ..., s_l\}.$$

Con esto queda demostrado el lema. Falta probar que $\mathcal{L}(M) = \mathcal{L}(M')$.

$$x \in \mathcal{L}(M)$$

$$\iff$$

$$\delta(q_0, x) = \{r_1, ..., r_k\} \land \{r_1, ..., r_k\} \cap F \neq \emptyset$$

$$\underset{\text{por lema anterior}}{\iff}$$

$$\delta'(q'_0, x) = [r_1, ..., r_k] \land [r_1, ..., r_k] \cap F' \neq \emptyset$$

$$\iff$$
$$x \in \mathcal{L}(M')$$

Concluimos, $\mathcal{L}(M) = \mathcal{L}(M')$.

□

2.5. Gramáticas regulares

2.5.1. Todo lenguaje generado por una gramática reg., tiene un AFND que lo reconoce

Dada una gramática regular $G = \langle V_N, V_T, P, S \rangle$, existe un AFND $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ tal que $\mathcal{L}(G) = \mathcal{L}(M)$. **← Teorema**

Demostración:
Definimos M tal que:

- $Q = \{q_A : A \in V_N\} \cup \{q_f\}$. ← Los estados son los no terminales + un estado q_f.
- $\Sigma = V_T$. ← El alfabeto de entrada es el conjunto de símbolos terminales.
- $\delta(q_A, a)$ con $A, B \in V_N$ y $a \in V_T$.
  - $q_B \in \delta(q_A, a) \Leftrightarrow (A \to aB) \in P$.
  - $q_f \in \delta(q_A, a) \Leftrightarrow (A \to a) \in P$.
- $q_0 = q_S$. ← El estado inicial es el símbolo inicial.
- $F = \{q_A : A \in V_N, (A \to \lambda) \in P\} \cup \{q_f\}$. ← Todo estado con producción que deriva en λ y nuestro estado q_f.

Para probar que $\mathcal{L}(G) = \mathcal{L}(M)$, vamos a probar un lema:

$$\forall w \in V_T^*, \text{ si } A \overset{*}{\Rightarrow} wB \text{ entonces } q_B \in \hat{\delta}(q_A, w).$$

Demostración del lema:
Hacemos inducción en la longitud de w.

Caso base: $|w| = 0$, i.e., $w = \lambda$.

Si $A \overset{*}{\Rightarrow} \lambda B$, entonces $A = B$ y $q_B \in \delta(q_A, \lambda)$, por definición de δ.
Luego, el caso base es verdadero.

Paso inductivo: asumimos que vale para w tal que $|w| = n$, probemos que vale para $|w| = n + 1$.

Queremos probar que si

$$A \overset{*}{\Rightarrow} waB \text{ entonces } q_B \in \hat{\delta}(q_A, wa).$$

Asumiendo que si $A \overset{*}{\Rightarrow} wB$ entonces $q_B \in \hat{\delta}(q_A, w)$ es verdadero.

Entonces,

$$\begin{aligned} A \overset{*}{\Rightarrow} waB &\iff \exists C \in V_N : A \overset{*}{\Rightarrow} wC \wedge (C \to aB) \in P \\ &\underset{\textbf{HI}}{\Longrightarrow} \exists q_C \in Q : q_C \in \hat{\delta}(q_A, w) \wedge q_B \in \delta(q_C, a) \\ &\iff q_B \in \delta\left(\hat{\delta}(q_C, w), a\right) \underset{\textbf{def. de } \delta}{=} \hat{\delta}(q_A, wa). \end{aligned}$$

□

Falta probar que $\mathcal{L}(G) = \mathcal{L}(M)$. Sea $w \in V_T^*, a \in V_T$.

Caso cadena vacía:

$$\begin{aligned}\lambda \in \mathcal{L}(G) &\iff S \overset{*}{\Rightarrow} \lambda \\ &\iff (S \to \lambda) \in P \\ &\iff q_S \in F \\ &\iff \lambda \in \mathcal{L}(M).\end{aligned}$$

Caso cadena no vacía:

$$\begin{aligned}wa \in \mathcal{L}(G) &\iff S \overset{*}{\Rightarrow} wa \\ &\iff \Big(\exists A \in V_N : S \overset{*}{\Rightarrow} wA \wedge (A \to a) \in P\Big) \vee \Big(\exists B \in V_N : S \overset{*}{\Rightarrow} waB \wedge (B \to \lambda) \in P\Big) \\ &\iff \Big(\exists A \in V_N : S \overset{*}{\Rightarrow} wA \wedge q_f \in \delta(q_A, a)\Big) \vee \Big(\exists B \in V_N : S \overset{*}{\Rightarrow} waB \wedge q_B \in F\Big) \\ &\underset{\textbf{Lem}}{\Longrightarrow} \Big(\exists q_A \in Q : q_A \in \hat{\delta}(q_S, w) \wedge q_f \in \delta(q_A, a)\Big) \vee \Big(\exists q_B \in Q : q_B \in \hat{\delta}(q_S, wa) \wedge q_B \in F\Big) \\ &\iff \Big(\delta\Big(\hat{\delta}(q_S, w), a\Big) \underset{\textbf{def. de } \delta}{=} \hat{\delta}(q_S, wa) \cap F \neq \emptyset\Big) \vee \Big(\hat{\delta}(q_S, wa) \cap F \neq \emptyset\Big) \\ &\iff \hat{\delta}(q_S, wa) \cap F \neq \emptyset \\ &\iff wa \in \mathcal{L}(M).\end{aligned}$$

Con esto queda demostrado que $\mathcal{L}(G) = \mathcal{L}(M)$.

□

2.5.2. Todo AFD, tiene una gramática regular que lo genera

Dado un AFD $M = \langle Q, \Sigma, \delta, q_0, F\rangle$, existe una gramática regular $G = \langle V_N, V_T, P, S\rangle$ tal que $\mathcal{L}(M) = \mathcal{L}(G)$. **← Teorema**

Demostración:
Definimos la gramática $G = \langle V_N, V_T, P, S\rangle$, donde:
- $V_N = \{A_q : q \in Q\}$.
- $V_T = \Sigma$.
- P, con $A_p, A_q \in V_N$ y $a \in V_T$:
  - $(A_q \to aA_p) \in P \iff \delta(q, a) = p$.
  - $(A_q \to a) \in P \iff \delta(q, a) = p \wedge p \in F$.
  - $(S \to \lambda) \in P \iff q_0 \in F$.
- $S = A_{q_0}$.

Para probar que $\mathcal{L}(M) = \mathcal{L}(G)$, vamos a probar un lema:

$$\delta(p, w) = q \iff A_p \overset{*}{\Rightarrow} wA_q.$$

Demostración del lema:
Hacemos inducción en la longitud de w.

Caso base: $|w| = 0$, i.e., $w = \lambda$.

Para $w = \lambda$, es cierto que $\delta(p, \lambda) \underset{\textbf{def. de } \delta}{=} p \iff A_p \underset{\textbf{en particular, cero pasos}}{\overset{*}{\Rightarrow}} \lambda A_p$.

Paso inductivo: asumimos que vale para w tal que $|w| = n$, probemos que vale para $|w| = n + 1$.

Queremos probar que

$$\delta(p, wa) = q \iff A_p \overset{*}{\Rightarrow} waA_q.$$

Entonces,

$$\begin{aligned}
\delta(p, wa) = q &\iff \exists r \in Q : \delta(p, w) = r \wedge \delta(r, a) = q \\
&\iff \exists r \in Q : \delta(p, w) = r \wedge \left(A_r \to aA_q\right) \in P \\
&\underset{\textbf{HI}}{\iff} \exists A_r \in V_N : A_p \overset{*}{\Rightarrow} wA_r \wedge \left(A_r \to aA_q\right) \in P \\
&\iff A_p \overset{*}{\Rightarrow} waA_q.
\end{aligned}$$

□

Falta probar que $\mathcal{L}(M) = \mathcal{L}(G)$.

Caso cadena vacía:

$$\begin{aligned}
\lambda \in \mathcal{L}(M) &\iff q_0 \in F \\
&\iff (S \to \lambda) \in P \\
&\iff S \overset{*}{\Rightarrow} \lambda \\
&\iff \lambda \in \mathcal{L}(G).
\end{aligned}$$

Caso cadena no vacía:

$$\begin{aligned}
wa \in \mathcal{L}(M) &\iff \delta(q_0, wa) \in F \\
&\iff \exists p \in Q : \delta(q_0, w) = p \wedge \delta(p, a) \in F \\
&\iff \exists p \in Q : \delta(q_0, w) = p \wedge \left(A_p \to a\right) \in P \\
&\underset{\textbf{Lem}}{\iff} \exists A_p \in V_N : A_{q_0} \overset{*}{\Rightarrow} wA_p \wedge \left(A_p \to a\right) \in P \\
&\iff A_{q_0} \overset{*}{\Rightarrow} wa \\
&\iff wa \in \mathcal{L}(G).
\end{aligned}$$

Con esto queda demostrado que $\mathcal{L}(M) = \mathcal{L}(G)$.

□

3. Lenguajes regulares: lema de pumping y propiedades de clausura

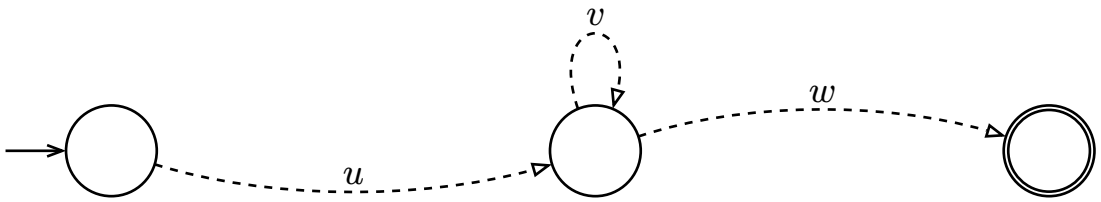
3.1. Lema de pumping para lenguajes regulares

3.1.1. Introducción

Si las longitudes de las cadenas de un lenguaje L están acotadas superiormente, por ejemplo: $x \in L \Rightarrow |x| \leq 20$, entonces L tiene que ser **finito**.

¿Qué condición tiene que cumplir el grafo de un autómata finito para que el lenguaje que éste acepta sea infinito?

Debe existir un camino desde el estado inicial hasta algún estado final que pase por algún ciclo.



El **lema de pumping** es una propiedad que cumplen todos los lenguajes regulares.
Eso quiere decir que si un lenguaje es regular, entonces cumple con el lema de pumping.
O dicho de otro modo, **si un lenguaje no cumple con el lema de pumping, entonces no es regular**.

3.1.2. Lema

Sea L un lenguaje regular. Existe una longitud n tal que para toda $z \in L$ con $|z| \geq n$ existe $u, v, w \in \Sigma^*$ tales que

$$z = uvw$$
$$|uv| \leq n$$
$$|v| \geq 1$$
$$\forall i \geq 0, uv^i w \in L.$$

3.1.3. Demostración

Para la demostración del lema de pumping, vamos a demostrar primero un lema sobre las transiciones entre configuraciones instantáneas.

Sea el AFD $M = \langle Q, \Sigma, \delta, q_0, F \rangle$.
Para todo $q \in Q$ y $x, y \in \Sigma^*$, si $(q, xy) \overset{*}{\vdash} (q, y)$ entonces $\forall i \geq 0, (q, x^i y) \overset{*}{\vdash} (q, y)$.

Es decir, si partiendo de un estado q, terminamos en el mismo estado q tras leer la cadena x, entonces hay un ciclo en el AFD que nos permite leer la cadena x cero o más veces.

Demostración del lema sobre transiciones entre configuraciones instantáneas:

Hacemos inducción en i, fijamos $q \in Q$ y $x \in \Sigma^*$; asumimos que $\forall y \in \Sigma^*$, $(q, xy) \overset{*}{\vdash} (q, y)$.

Caso base: $i = 0$.

Nos queda
$(q, xy) \overset{*}{\vdash} (q, y)$ entonces $(q, y) \overset{*}{\vdash} (q, y)$,
lo que al asumir el antecedente es trivialmente cierto.

Paso inductivo: asumimos que vale para i, probemos que vale para $i+1$.

$$(q, x^{i+1}y) \underset{\textbf{def}}{=} (q, xx^iy)$$

Luego, por asunción del antecedente,

$$(q, xx^iy) \overset{*}{\vdash} (q, x^iy)$$
$$\underset{\textbf{HI}}{\Longrightarrow} (q, x^iy) \overset{*}{\vdash} (q, y).$$

Concluimos que $(q, x^{i+1}y) \overset{*}{\vdash} (q, y)$.

□

Con este lema, podemos demostrar el lema de pumping.

Demostración del lema de pumping:

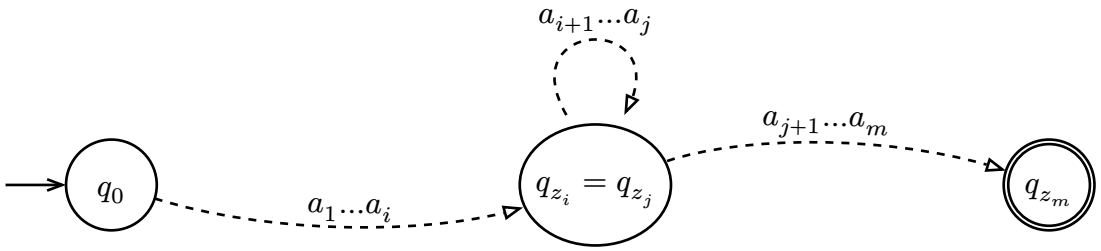
Sea AFD $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ tal que $\mathcal{L}(M) = L$. Sea n su cantidad de estados. Sea z una cadena de longitud $m \geq n$, $z = a_1...a_m$.

Para aceptar z, usamos m transiciones *(aristas del grafo)*, por lo tanto $m+1$ estados *(nodos del grafo)*.

Como $m + 1 > n$, para aceptar z, **el autómata pasa dos o más veces por un mismo estado al reconocer la cadena**.

Sea $q_{z_0}, q_{z_1}, ..., q_{z_m} \in Q$, con $q_{z_0} = q_0$ y q_{z_m} un estado final, la sucesión de estados desde q_0 hasta aceptar z.

Gráficamente,



Formalmente,
Existen i, j tales que $q_{z_i} = q_{z_j}$ con $0 \leq i < j \leq n$.

Luego, $z = uvw$, con

$$u = \begin{cases} a_1...a_i & \text{si } i{>}0 \\ \lambda & \text{si } i{=}0 \end{cases}$$

$$v = a_{i+1}...a_j$$

$$w = \begin{cases} a_{j+1}...a_m & \text{si } j{<}m \\ \lambda & \text{si } j{=}m \end{cases}$$

Es fácil ver que $|uv| \leq n$ y $|v| \geq 1$.
Pues,

$$|uv| = i + (j - i) = j \leq n \text{ es verdadero por } 0 \leq i < j \leq n.$$

$|v| = j - i \geq 1 \iff i + 1 \leq j \iff i < j$ es verdadero por $0 \leq i < j \leq n$.

A su vez,
$(q_0, uvw) \overset{*}{\vdash} \left(q_{z_i}, vw\right) \overset{*}{\vdash} \left(q_{z_j}, w\right) \overset{*}{\vdash} \left(q_{z_m}, \lambda\right)$,
es verdadero por construcción.

Pero $q_{z_i} = q_{z_j}$. Finalmente,

Como $\left(q_{z_i}, vw\right) \overset{*}{\vdash} \left(q_{z_j}, w\right)$,

por el lema sobre transiciones entre configuraciones instantáneas,

podemos decir que $\forall i \geq 0, \left(q_{z_i}, v^i w\right) \overset{*}{\vdash} \left(q_{z_j}, w\right)$,

y sabemos que $\left(q_{z_j}, w\right) \overset{*}{\vdash} \left(q_{z_m}, \lambda\right)$.

Por lo que concluimos que $\forall i \geq 0, uv^i w \in L$.

□

3.1.4. Algunas demostraciones donde se aplica el lema de pumping

Sea AFD $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ con $|Q| = n$, tal que $\mathcal{L}(M) = L$.

3.1.4.1. $L \neq \emptyset \iff \exists x \in \Sigma^* : \hat{\delta}(q_0, x) \in F \wedge |x| < n$

$\Rightarrow$) Como $L \neq \emptyset$, existe $x \in L$. Por el lema de pumping, sé que existe un m tal que para toda cadena $|z| \geq m$, se puede descomponer $z = uvw$ con $|uv| \leq n$ y $|v| \geq 1$, de manera tal que $\forall i \geq 0, uv^i w \in L$.

Propongo $z = x$.

- Supongamos que $|x| < n$.
  Sé que $\hat{\delta}(q_0, x) \in F$ por hipótesis ($L \neq \emptyset$).
  Y $|x| < n$ por el caso.
- Supongamos que $|x| \geq n$.
  Por el lema de pumping, sé que puedo descomponer $x = uvw$ con $|uv| \leq n$ y $|v| \geq 1$, de manera tal que $\forall i \geq 0, uv^i w \in L$.
  Ya que al tener longitud $\geq n$, pasa por un ciclo *(ver demostración del lema de pumping).*
  Asumiendo dada la descomposición, por el lema de pumping, puedo tomar $i = 0$.
  Luego, $|uw| \underset{\text{pues } |v| \geq 1}{<} |uvw| = |x| \leq n \implies |uw| < n$, y $uw \in L$ por el lema de pumping.

□

$\Leftarrow$) $\exists x \in \Sigma^* : \hat{\delta}(q_0, x) \in F \wedge |x| < n \Rightarrow \exists x \in \Sigma^* : \hat{\delta}(q_0, x) \in F \Rightarrow x \in L \Rightarrow L \neq \emptyset$.
Dicho de otro modo, es trivial.

□

3.1.4.2. L es infinito $\iff \exists x \in \Sigma^* : \hat{\delta}(q_0, x) \in F \wedge n \leq |x| < 2n$

$\Rightarrow$) Como L es infinito, sé particularmente que $L \neq \emptyset$, por lo que $\exists x \in \Sigma^* : \hat{\delta}(q_0, x) \in F$.
Quiero ver que, en particular ese x cumple con $n \leq |x| < 2n$.

Intuitivamente, si L es infinito, entonces tiene que haber un ciclo en el AFD, pues debe haber algún lugar donde se esten «generando» cadenas infinitas.
Esto me permite afirmar que hay al menos una cadena que cumple con $|x| \geq n$.
Me falta ver que al menos una cadena cumple con $|x| < 2n$.

Supongamos que no hay ninguna cadena en L que cumpla con $n \leq |x| < 2n$.
Sin pérdida de generalidad, sea $|x| = 2n$.
En caso de que $|x| > 2n$, se puede repetir el argumento de la demostración cuantas veces sea necesario. Si te queda x tal que $|x| < 2n$, llegaste al absurdo que buscábamos, si te queda x tal que $|x| \geq 2n$, podés bombear x otra vez pues es $\geq n$, luego, repetís el argumento.

Luego, por el lema de pumping, como $|x| \geq n$, puedo descomponer $x = uvw$ con $|uv| \leq n$ y $|v| \geq 1$, de manera tal que $\forall i \geq 0, uv^iw \in L$.
Por lo que $|uw| \in L$.

Como $|uvw| = 2n$ y $1 \leq |v| \leq n$, tenemos que

$$n \underbrace{\leq}_{\text{como mucho le saqué n}} |uw| \underbrace{<}_{\text{como mínimo le saqué 1}} 2n.$$

Contradiciendo que no hay ninguna cadena en L que cumpla con $n \leq |x| < 2n$.

□

$\Leftarrow$) Supongo que $\exists x \in \Sigma^* : \hat{\delta}(q_0, x) \in F \land n \leq |x| < 2n$.
Por el lema de pumping, puedo descomponer $x = uvw$ con $|uv| \leq n$ y $|v| \geq 1$, de manera tal que $\forall i \geq 0, uv^iw \in L$, esto último implica que L es infinito.

□

3.1.5. Propiedades de clausura de los lenguajes regulares

3.1.5.1. Unión

El conjunto de lenguajes regulares incluido en Σ^* es cerrado respecto de la unión.

3.1.5.1.1. Demostración

Sean L_1 y L_2 lenguajes regulares. Quiero probar que $L_1 \cup L_2$ es un lenguaje regular.

Sean $M_1 = \langle Q_1, \Sigma, \delta_1, q_{1_0}, F_1 \rangle$ y $M_2 = \langle Q_2, \Sigma, \delta_2, q_{2_0}, F_2 \rangle$ dos AFD, donde $Q_1 \cap Q_2 = \emptyset$ tales que $\mathcal{L}(M_1) = L_1$ y $\mathcal{L}(M_2) = L_2$.

Defino AFD $M = \langle Q_1 \times Q_2, \Sigma, \delta, \left(q_{1_0}, q_{2_0}\right), F \rangle$, donde

- $\forall q_1 \in Q_1, \forall q_2 \in Q_2, \forall a \in \Sigma, \delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
- $F = \{(f_1, f_2) : f_1 \in F_1 \vee f_2 \in F_2\}$

tal que $\mathcal{L}(M) = L$.

Para probar que $L = L_1 \cup L_2$, basta probar que: $x \in \mathcal{L}(M) \iff x \in \mathcal{L}(M_1) \vee x \in \mathcal{L}(M_2)$.

$$
\begin{aligned}
x \in \mathcal{L}(M) &\iff \delta\Big(\big(q_{1_0}, q_{2_0}\big), x\Big) \in F \\
&\iff \Big(\delta_1\big(q_{1_0}, x\big), \delta_2\big(q_{2_0}, x\big)\Big) \in F \\
&\iff \delta_1\big(q_{1_0}, x\big) \in F_1 \vee \delta_2\big(q_{2_0}, x\big) \in F_2 \\
&\iff x \in \mathcal{L}(M_1) \vee x \in \mathcal{L}(M_2).
\end{aligned}
$$

□

3.1.5.2. Concatenación

El conjunto de lenguajes regulares incluido en Σ^* es cerrado respecto de la concatenación.

3.1.5.2.1. Demostración

Sean L_1 y L_2 lenguajes regulares. Quiero probar que $L_1 L_2$ es un lenguaje regular.

Sean $M_1 = \langle Q_1, \Sigma, \delta_1, q_{1_0}, F_1\rangle$ y $M_2 = \langle Q_2, \Sigma, \delta_2, q_{2_0}, F_2\rangle$ AFDs, donde $Q_1 \cap Q_2 = \emptyset$ tales que $\mathcal{L}(M_1) = L_1$ y $\mathcal{L}(M_2) = L_2$.

Defino AFND-λ $M = \langle Q_1 \cup Q_2, \Sigma \cup \{\lambda\}, \delta, q_{1_0}, F_2\rangle$, donde

- $\forall q \in Q_1, \forall a \in \Sigma, \delta_1(q, a) \in \delta(q, a)$
- $\forall q \in Q_2, \forall a \in \Sigma, \delta_2(q, a) \in \delta(q, a)$
- $\forall f \in F_1, \forall a \in \Sigma : q_{2_0} \in \delta_1(f, \lambda)$

Ahora falta probar que $\mathcal{L}(M) = L_1 L_2$.
Para ello probamos que $x_1 x_2 \in \mathcal{L}(M) \iff x_1 \in \mathcal{L}(M_1) \wedge x_2 \in \mathcal{L}(M_2)$.

Sean $x_1, x_2 \in \Sigma^*$.

$$
\begin{aligned}
x_1 x_2 \in \mathcal{L}(M) &\iff \delta\big(q_{1_0}, x_1 x_2\big) \in F_2 \\
&\iff \delta_1\big(q_{1_0}, x_1\big) \in F_1 \wedge q_{2_0} \in Cl_\lambda(f) \wedge \delta_2\big(q_{2_0}, x_2\big) \in F_2 \\
&\iff \delta_1\big(q_{1_0}, x_1\big) \in F_1 \wedge \delta_2\big(q_{2_0}, x_2\big) \in F_2 \\
&\iff x_1 \in \mathcal{L}(M_1) \wedge x_2 \in \mathcal{L}(M_2).
\end{aligned}
$$

□

3.1.5.3. Complemento

El conjunto de lenguajes regulares incluido en Σ^* es cerrado respecto al complemento.

3.1.5.3.1. Demostración

Sea $L \subseteq \Delta^*$ regular, con $\Delta \subseteq \Sigma$.
Sea $M = \langle Q, \Delta, \delta, q_0, F\rangle$ tal que $\mathcal{L}(M) = L$ y δ está definida para todos los elementos del alfabeto Δ.

El AFD $M' = \langle Q, \Delta, \delta, q_0, Q - F\rangle$ acepta $\Delta^* - \mathcal{L}(M)$.
Es decir, $\mathcal{L}(M') = \Delta^* - \mathcal{L}(M)$.

En caso de que $\Delta = \Sigma$, $\overline{\mathcal{L}(M)} = \mathcal{L}(M') \cup \Sigma^*(\Sigma - \Delta)\Sigma^*$,
que es la unión de dos lenguajes regulares y por lo tanto regular.

□

3.1.5.4. Intersección

El conjunto de lenguajes regulares incluido en Σ^* es cerrado respecto de la intersección.

3.1.5.4.1. Demostración

Dados M_1 y M_2 AFDs tales que $\mathcal{L}(M_1) = L_1$ y $\mathcal{L}(M_2) = L_2$, el autómata M que aceptará el lenguaje $L_1 \cap L_2$, estará dado por $M = \langle Q_1 \times Q_2, \Sigma, \delta, \left(q_{1_0}, q_{2_0}\right), F\rangle$ con:

- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ para $q_1 \in Q_1$ y $q_2 \in Q_2$.
- $F = F_1 \times F_2 = \{(f_1, f_2) : f_1 \in F_1 \wedge f_2 \in F_2\}$.

Luego,

$$\begin{aligned} x \in \mathcal{L}(M) &\iff \delta\Big(\left(q_{1_0}, q_{2_0}\right), x\Big) \in F \\ &\iff \Big(\delta_1\left(q_{1_0}, x\right), \delta_2\left(q_{2_0}, x\right)\Big) \in F \\ &\iff \delta_1\left(q_{1_0}, x\right) \in F_1 \wedge \delta_2\left(q_{2_0}, x\right) \in F_2 \\ &\iff x \in \mathcal{L}(M_1) \wedge x \in \mathcal{L}(M_2). \end{aligned}$$

□

Otra forma de demostrarlo es sabiendo que el conjunto de lenguajes regulares incluido en Σ^* es cerrado respecto de la unión y complementación.
Para ello, aplicamos las leyes de De Morgan.

Dados L_1 y L_2 lenguajes regulares,

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} = \overline{\overline{L_1} \cup \overline{L_2}}.$$

□

3.1.5.5. Conclusión de las propiedades de clausura anteriores

De los teoremas anteriores podemos afirmar que el conjunto de los lenguajes regulares incluidos en Σ^* es un álgebra booleana de conjuntos.

3.1.5.6. Reversa

El conjunto de lenguajes regulares incluido en Σ^* es cerrado respecto de la reversa.

Falta la demostración.

3.1.5.7. Union e intersección finita

La unión e intersección finita de lenguajes regulares dan por resultado un lenguaje regular.

3.1.5.7.1. Demostración

Debemos ver que,

$$\begin{gathered} \forall n \in \mathbb{N}, \bigcup_{i=1}^{n} L_i \text{ es regular,} \\ \text{y} \\ \forall n \in \mathbb{N}, \bigcap_{i=1}^{n} L_i \text{ es regular.} \end{gathered}$$

Por inducción en n:

Caso base: $n = 0$.

$\bigcup_{i=1}^{0} L_i = \emptyset$ y $\bigcap_{i=1}^{0} L_i = \emptyset$, que son lenguajes regulares.

Caso inductivo: supongamos que vale para n, probemos que vale para $n + 1$.

$\bigcup_{i=1}^{n+1} L_i = \bigcup_{i=1}^{n} L_i \cup L_{n+1}$, que es un lenguaje regular por la propiedad de clausura de la unión.

$\bigcap_{i=1}^{n+1} L_i = \bigcap_{i=1}^{n} L_i \cap L_{n+1}$, que es un lenguaje regular por la propiedad de clausura de la intersección.

□

- **Observación:** los lenguajes regulares no están clausurados por unión infinita.

Ejemplo clásico, para cada $i \geq 1$, $L_i = \{a^i b^i\}$ es regular.
Sin embargo, $\bigcup_{i=1}^{\infty} L_i = \bigcup_{i=1}^{\infty} \{a^i b^i\} = \{a^n b^n : n \in \mathbb{N}\}$ no es regular.

3.1.5.8. Todo lenguaje finito es regular

3.1.5.8.1. Demostración

Sea L un lenguaje finito, con n cadenas, $L = \{\alpha_1, \alpha_2, ..., \alpha_n\}$.

Para cada $i = 1, 2, ..., n$, sea $L_i = \{\alpha_i\}$. Cada $L_i = \{\alpha_i\}$ es regular.

Entonces, $L = \bigcup_{i=1}^{n} L_i = \bigcup_{i=1}^{n} \{\alpha_i\}$, que es un lenguaje regular, pues vimos que la unión finita de lenguajes regulares es regular.

4. Expresiones regulares

4.1. Definición

Dado un alfabeto Σ, una expresión regular denota un lenguaje sobre Σ:

- $\emptyset$ es una expresión regular que denota el conjunto vacío $\emptyset$.
- λ es una expresión regular que denota el conjunto $\{\lambda\}$.
- para cada $a \in \Sigma$, a es una expresión regular que denota el conjunto $\{a\}$.
- si r y s denotan los lenguajes R y S entonces
  - $r \mid s$ denota $R \cup S$,
  - rs denota RS,
  - r^* denota R^*, y
  - r^+ denota R^+.

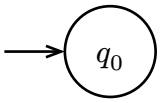
4.2. Todo lenguaje descrito por una expresión regular, tiene un autómata finito que lo acepta

Dada una expresión regular r, existe un AFND-λ M con un solo estado final y sin transiciones a partir del mismo tal que $\mathcal{L}(M) = \mathcal{L}(r)$. **← Teorema**

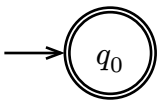
4.2.1. Demostración

Casos base:

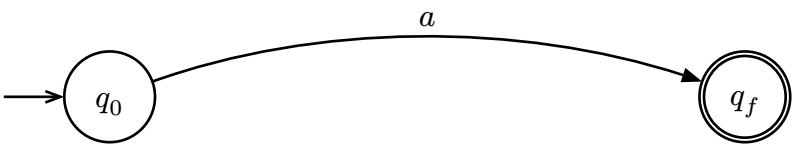
$r = \emptyset$



$r = \lambda$



$r = a$



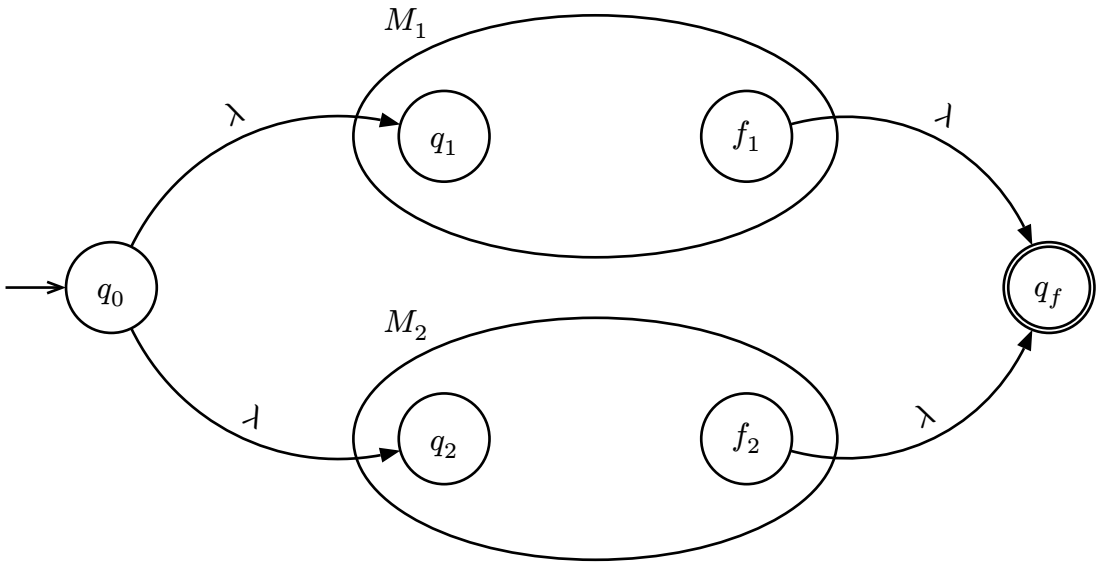
Paso inductivo:

Supongamos que la expresión regular es $r_1 \mid r_2$, $r_1 r_2$, r_1^*, ó r_2^+ y asumimos que vale la propiedad para r_1 y para r_2.
Es decir, tanto para r_1 como para r_2 existen AFND-λ M_1 y M_2 con un solo estado final y sin transiciones a partir del mismo, tal que $\mathcal{L}(M_1) = \mathcal{L}(r_1)$ y $\mathcal{L}(M_2) = \mathcal{L}(r_2)$.

$r = r_1 \mid r_2$

Por **H.I.**, existen $M_1 = \langle Q_1, \Sigma_1, \delta_1, q_1, \{f_1\}\rangle$ y $M_2 = \langle Q_2, \Sigma_2, \delta_2, q_2, \{f_2\}\rangle$ con un solo estado final y sin transiciones a partir del mismo, tales que $\mathcal{L}(M_1) = \mathcal{L}(r_1)$ y $\mathcal{L}(M_2) = \mathcal{L}(r_2)$.

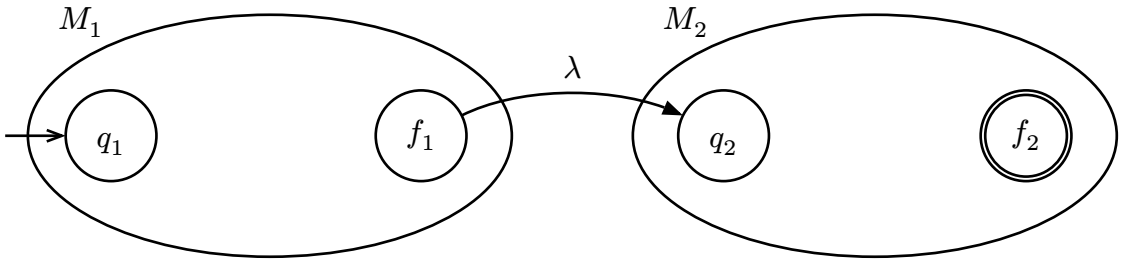
Sea $M = \langle Q_1 \cup Q_2 \cup \{q_0, q_f\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{q_f\}\rangle$.



- $\delta(q_0, \lambda) = \{q_1, q_2\}$.
- $\delta(q, a) = \delta_1(q, a)$ para $q \in Q_1 - \{f_1\}$ y $a \in \Sigma_1 \cup \{\lambda\}$.
- $\delta(q, a) = \delta_2(q, a)$ para $q \in Q_2 - \{f_2\}$ y $a \in \Sigma_2 \cup \{\lambda\}$.
- $\delta(f_1, \lambda) = \delta(f_2, \lambda) = \{q_f\}$.

$r = r_1 r_2$

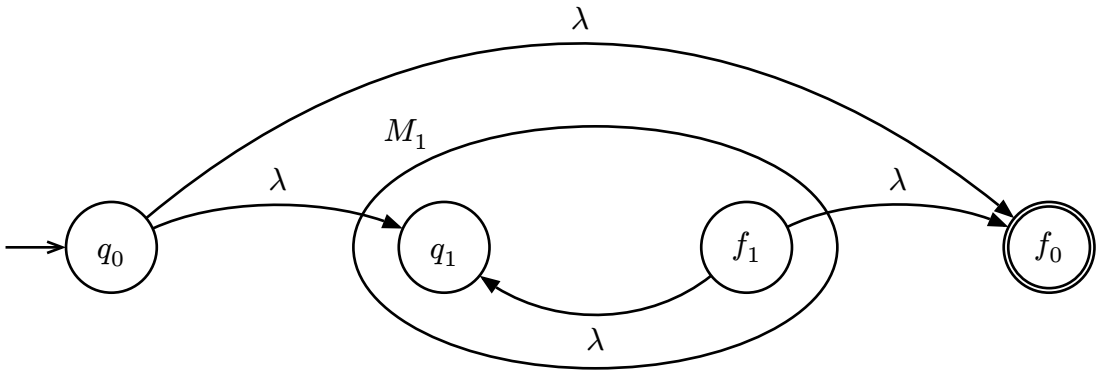
Por **H.I.** existen $M_1 = \langle Q_1, \Sigma_1, \delta_1, q_1, \{f_1\}\rangle$ y $M_2 = \langle Q_2, \Sigma_2, \delta_2, q_2, \{f_2\}\rangle$ con un solo estado final y sin transiciones a partir del mismo, tales que $\mathcal{L}(M_1) = \mathcal{L}(r_1)$ y $\mathcal{L}(M_2) = \mathcal{L}(r_2)$.
Entones, podemos construir el autómata $M = \langle Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, q_1, \{f_2\}\rangle$.



- $\delta(q, a) = \delta_1(q, a)$ para $q \in Q_1 - \{f_1\}$ y $a \in \Sigma_1 \cup \{\lambda\}$.
- $\delta(f_1, \lambda) = \{q_2\}$.
- $\delta(q, a) = \delta_2(q, a)$ para $q \in Q_2 - \{f_2\}$ y $a \in \Sigma_2 \cup \{\lambda\}$.

$r = r_1^*$

Por **H.I.** existe $M_1 = \langle Q_1, \Sigma_1, \delta_1, q_1, \{f_1\}\rangle$ con un solo estado final y sin transiciones a partir del mismo, tal que $\mathcal{L}(M_1) = \mathcal{L}(r_1)$.
Entonces, podemos construir el autómata $M = \langle Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\}\rangle$.



- $\delta(q, a) = \delta_1(q, a)$ para $q \in Q_1 - \{f_1\}$ y $a \in \Sigma_1 \cup \{\lambda\}$.
- $\delta(q_0, \lambda) = \delta(f_1, \lambda) = \{q_1, f_0\}$.

$r = r_1^+$

Dado que $r_1^+ = r_1 r_1^*$, queda demostrado por los casos anteriores.

□

4.3. Todo lenguaje aceptado por un autómata finito, tiene una expresión regular que lo describe

Dado un AFD $M = \langle Q, \Sigma, \delta, q_1, F\rangle$, existe una expresión regular r tal que $\mathcal{L}(M) = \mathcal{L}(r)$. **← Teorema**

4.3.1. Demostración

Denoetemos con $R_{i,j}^k$ el conjunto de cadenas de Σ^* que llevan al autómata M desde el estado q_i al estado q_j pasando por estados cuyo índice es, a lo sumo, k.
Definamos $R_{i,j}$ en forma recursiva:

$$R_{i,j}^0 = \begin{cases} \{a : \delta(q_i, a) = q_j\} & \text{si } i \neq j \\ \{a : \delta(q_i, a) = q_j\} \cup \{\lambda\} & \text{si } i = j \end{cases}$$

$$R_{i,j}^k = R_{i,k}^{k-1}\left(R_{k,k}^{k-1}\right)^* R_{k,j}^{k-1} \cup R_{i,j}^{k-1} \text{ para } k \in \mathbb{N} \geq 1$$

Demostremos que para todo $R_{i,j}^k$, existe una expresión regular $r_{i,j}^k$ tal que $\mathcal{L}\left(r_{i,j}^k\right) = R_{i,j}^k$.

Caso base: $k = 0$.
Debemos dar $r_{i,j}^0$ tal que $\mathcal{L}\left(r_{i,j}^0\right) = R_{i,j}^0$.
$R_{i,j}$ es el conjunto de cadenas de un solo caracter o λ.

Por lo tanto, $r_{i,j}^0$ es:

- $a_1 \mid ... \mid a_p$
  - con $a_1, ..., a_p$ símbolos de Σ, si $\delta(q_i, a_s) = q_j$ para $s = 1, ..., p$ y $q_i \neq q_j$.
- $a_1 \mid ... \mid a_p \mid \lambda$
  - con $a_1, ..., a_p$ símbolos de Σ, si $\delta(q_i, a_s) = q_j$ para $s = 1, ..., p$ y además $q_i = q_j$.
- $\emptyset$, si no existe ningún a_i que una q_i y q_j y $q_i \neq q_j$.
- λ, si no existe ningún a_i que una q_i y q_j y además $q_i = q_j$.

Paso inductivo:

Por **H.I.** tenemos:

$\mathcal{L}\left(r_{i,k}^{k-1}\right) = R_{i,k}^{k-1} \qquad \mathcal{L}\left(r_{k,k}^{k-1}\right) = R_{k,k}^{k-1} \qquad \mathcal{L}\left(r_{k,j}^{k-1}\right) = R_{k,j}^{k-1} \qquad \mathcal{L}\left(r_{i,j}^{k-1}\right) = R_{i,j}^{k-1}$

Definimos $r_{i,j}^{k} = r_{i,k}^{k-1}\left(r_{k,k}^{k-1}\right)^{*} r_{k,j}^{k-1} \mid r_{i,j}^{k-1}$ y verificamos que:

$$\begin{aligned}
\mathcal{L}\left(r_{i,j}^{k}\right) &= \mathcal{L}\left(r_{i,k}^{k-1}\left(r_{k,k}^{k-1}\right)^{*} r_{k,j}^{k-1} \mid r_{i,j}^{k-1}\right) \\
&= \mathcal{L}\left(r_{i,k}^{k-1}\left(r_{k,k}^{k-1}\right)^{*} r_{k,j}^{k-1}\right) \cup \mathcal{L}\left(r_{i,j}^{k-1}\right) \\
&= \mathcal{L}\left(r_{i,k}^{k-1}\right)\mathcal{L}\left(\left(r_{k,k}^{k-1}\right)^{*}\right)\mathcal{L}\left(r_{k,j}^{k-1}\right) \cup \mathcal{L}\left(r_{i,j}^{k-1}\right) \\
&= \mathcal{L}\left(r_{i,k}^{k-1}\right)\left(\mathcal{L}\left(r_{k,k}^{k-1}\right)\right)^{*}\mathcal{L}\left(r_{k,j}^{k-1}\right) \cup \mathcal{L}\left(r_{i,j}^{k-1}\right) \\
&= R_{i,k}^{k-1}\left(R_{k,k}^{k-1}\right)^{*} R_{k,j}^{k-1} \cup R_{i,j}^{k-1} \\
&= R_{i,j}^{k}.
\end{aligned}$$

Entonces, como $Q = \{q_1, ..., q_n\}$ y q_1 es el estado inicial de M,

$$\begin{aligned}
\mathcal{L}(M) &= R_{1_{j_1}}^{n} \cup ... \cup R_{1_{j_m}}^{n}, \text{ con } F = \left\{q_{j_1}, ..., q_{j_m}\right\} \\
&= \mathcal{L}\left(r_{1_{j_1}}^{n}\right) \cup ... \cup \mathcal{L}\left(r_{1_{j_m}}^{n}\right) \\
&= \mathcal{L}\left(r_{1_{j_1}}^{n} \mid ... \mid r_{1_{j_m}}^{n}\right).
\end{aligned}$$

Concluimos que $\mathcal{L}(M) = \mathcal{L}\left(r_{1_{j_1}}^{n} \mid ... \mid r_{1_{j_m}}^{n}\right)$.

□

5. Autómatas de pila y gramáticas libres de contexto

5.1. Autómatas de pila

Un **autómata de pila** (AP) $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ es un modelo computacional definido formalmente como un sistema de siete componentes:

- Q es un conjunto finito de estados.
- Σ es un alfabeto finito de entrada.
- Γ es un alfabeto finito de la pila.
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ es la función de transición:
  - $\delta(q, a, Z) = \{(p_1, \gamma_1), ..., (p_m, \gamma_m)\}$.
- $q_0 \in Q$ es el estado inicial.
- $Z_0 \in \Gamma$ es la configuración inicial de la pila.
- $F \subseteq Q$ es el conjunto de estados finales.

La interpretación de $\delta(q, x, Z) = \{(p_1, \gamma_1), ..., (p_n, \gamma_n)\}$, con $q, p_1, ..., p_n \in Q$, $x \in (\Sigma \cup \{\lambda\})$, $Z \in \Gamma$, y $\gamma_i \in \Gamma^*$ es la siguiente.

Cuando el estado del autómata es q, el símbolo que la cabeza lectora está inspeccionando en ese momento es x, y en el tope de la pila nos encontramos el símbolo Z, se realizan las siguientes acciones:

1. Si $x \in \Sigma$, es decir no es la cadena vacía, la cabeza lectora avanza una posición para inspeccionar el siguiente símbolo.
2. Se elimina el símbolo Z de la pila del autómata.
3. Se selecciona un par (p_i, γ_i) entre los existentes en la definición de $\delta(q, x, Z)$.
4. Se apila la cadena $\gamma_i = c_1 c_2 ... c_k$, con $c_i \in \Gamma$ en la pila del autómata, quedando el símbolo c_1 en el tope de la pila.
5. Se cambia el control del autómata al estado p_i.

5.1.1. Autómatas de pila determinísticos

Un autómata de pila $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ es determinístico si para todo $q \in Q$, $x \in \Sigma$, $Z \in \Gamma$ se cumplen las siguientes condiciones:

1. $\#\delta(q, x, Z) \leq 1$.
2. $\#\delta(q, \lambda, Z) \leq 1$.
3. si $\#\delta(q, \lambda, Z) = 1$ entonces $\#\delta(q, x, Z) = 0$.

5.1.2. La no equivalencia entre los APD y los APND

No es cierto que para cada autómata de pila no determinístico existe otro determinístico que reconoce el mismo lenguaje.

Esto se demuestra con un contraejemplo.
$L = \{\omega\omega^R\}$, donde ω^R es la cadena ω invertida, es aceptado por un APND, pero no es aceptado por ningún APD.

La intuición detrás de esto es que para reconocer $\omega\omega^R$, el AP debería saber cuando leyó completamente ω, y recordarla, para luego compararla con ω^R.
Sin embargo, ¿cuándo sabe el AP que terminó de leer ω?

En un APND, el autómata puede bifurcarse en varios caminos, hasta encontrar el corte correcto en $\omega\omega^R$.
En un APD, el autómata debería adivinar cuál es el corte correcto, lo cual no es posible.

5.1.3. Configuración de un autómata de pila

Sea un autómata de pila $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$.
Una **configuración** de M es una terna (q, ω, γ), donde:

- $q \in Q$ es el estado actual del autómata.
- $\omega \in \Sigma^*$ es la parte de la cadena de entrada que falta procesar.
- $\gamma \in \Gamma^*$ es el contenido de la pila.

La **configuración inicial** del autómata para la cadena ω_0 es (q_0, ω_0, Z_0).

5.1.4. Cambio de configuración $\vdash$

Sea un autómata de pila $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$.
Para todo $x \in \Sigma, \omega \in \Sigma^*, Z \in \Gamma, \gamma, \pi \in \Gamma^*, q, q' \in Q$:

- $(q, x\omega, Z\pi) \vdash (q', \omega, \gamma\pi)$ si $(q', \gamma) \in \delta(q, x, Z)$.
- $(q, \omega, Z\pi) \vdash (q', \omega, \gamma\pi)$ si $(q', \gamma) \in \delta(q, \lambda, Z)$.

5.1.5. Lenguaje aceptado por un autómata de pila

Sea un autómata de pila $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$.

El lenguaje aceptado por M por **estado final** es:

$$\mathcal{L}(M) = \left\{\omega \in \Sigma^* : \exists f \in F, \exists \gamma \in \Gamma^* \text{ que cumplen } (q_0, \omega, Z_0) \overset{*}{\vdash} (f, \lambda, \gamma)\right\}.$$

El lenguaje aceptado por M por **pila vacía** es:

$$\mathcal{L}_\lambda(M) = \left\{\omega \in \Sigma^* : \exists p \in Q \text{ que cumple } (q_0, \omega, Z_0) \overset{*}{\vdash} (p, \lambda, \lambda)\right\}.$$

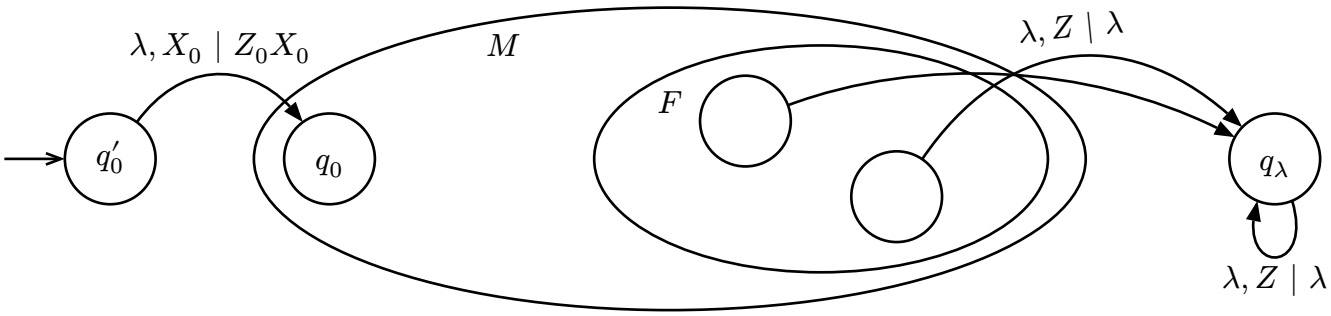
5.1.6. Los autómatas de pila no determinísticos por pila vacía y por estado final son equivalentes

Para cada **APND** M existe un **APND** M', tal que:

$$\mathcal{L}(M) = \mathcal{L}_\lambda(M')$$

Demostración:

Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ un **APND**.
Definimos el **APND** $M' = \langle Q \cup \{q_\lambda, q_0'\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q_0', X_0, \emptyset\rangle$:



Formalmente,

1. $\delta'(q'_0, \lambda, X_0) = \{(q_0, Z_0X_0)\}$. ← M' entra al estado inicial de M con Z_0X_0 en la pila, así evita la pila vacía.
2. $\forall q \in Q, \forall x \in \Sigma \cup \{\lambda\}, \forall Z \in \Gamma, \delta'(q, x, Z) = \delta(q, x, Z)$ ← M' mantiene las transiciones de M.
3. $\forall f \in F, \forall Z \in \Gamma \cup \{X_0\}, (q_\lambda, \lambda) \in \delta'(f, \lambda, Z)$ ← Si M llega a un estado final, lo mandamos a vaciar la pila.
4. $\forall Z \in \Gamma \cup \{X_0\}, \delta'(q_\lambda, \lambda, Z) = \{(q_\lambda, \lambda)\}$ ← Vamos vaciamos la pila de M'.

Queremos ver que $\mathcal{L}(M) = \mathcal{L}_\lambda(M')$.
Para ello veamos que $\mathcal{L}(M) \subseteq \mathcal{L}_\lambda(M')$ y $\mathcal{L}(M) \supseteq \mathcal{L}_\lambda(M')$.

Si $\omega \in \mathcal{L}(M)$, entonces $(q_0, \omega, Z_0) \overset{*}{\underset{M}{\vdash}} (f, \lambda, \gamma)$, con $f \in F, \gamma \in \Gamma^*$.

Luego, como $\delta'(q'_0, \lambda, X_0) \underset{\text{definición de } \delta'}{=} \{(q_0, Z_0X_0)\}$ sabemos que

$$(q'_0, \omega, X_0) \underset{M'}{\vdash} (q_0, \omega, Z_0X_0)$$

se cumple.

A su vez, sabemos que $\forall q \in Q, \forall x \in \Sigma \cup \{\lambda\}, \forall Z \in \Gamma, \delta'(q, x, Z) \underset{\text{definición de } \delta'}{=} \delta(q, x, Z)$ es cierto, lo que nos permite afirmar que

$$(q_0, \omega, Z_0X_0) \overset{*}{\underset{M'}{\vdash}} (f, \lambda, \gamma X_0).$$

Por otro lado, como $\forall f \in F, \forall Z \in \Gamma \cup \{X_0\}, (q_\lambda, \lambda) \underset{\text{definición de } \delta'}{\in} \delta'(f, \lambda, Z)$, y
$\forall Z \in \Gamma \cup \{X_0\}, \delta'(q_\lambda, \lambda, Z) \underset{\text{definición de } \delta'}{=} \{(q_\lambda, \lambda)\}$ sabemos que

$$(f, \lambda, \gamma X_0) \overset{*}{\underset{M'}{\vdash}} (q_\lambda, \lambda, \lambda).$$

Por lo tanto, $(q'_0, \omega, X_0) \overset{*}{\underset{M'}{\vdash}} (q_\lambda, \lambda, \lambda)$, lo que implica que $\omega \in \mathcal{L}_\lambda(M')$.

Veamos que $\mathcal{L}(M) \supseteq \mathcal{L}_\lambda(M')$.

Si $\omega \in \mathcal{L}_\lambda(M')$, existe la secuencia

$$(q'_0, \omega, X_0) \underset{M'}{\vdash} \underbrace{(q_0, \omega, Z_0X_0) \overset{*}{\underset{M'}{\vdash}} (f, \lambda, \gamma X_0)}_{A} \overset{*}{\underset{M'}{\vdash}} (q_\lambda, \lambda, \lambda).$$

Pero la transición A implica

$$(q_0, \omega, Z_0) \overset{*}{\underset{M}{\vdash}} (f, \lambda, \gamma).$$

Por lo que concluimos que, si $\omega \in \mathcal{L}_\lambda(M')$, entonces $\omega \in \mathcal{L}(M)$.

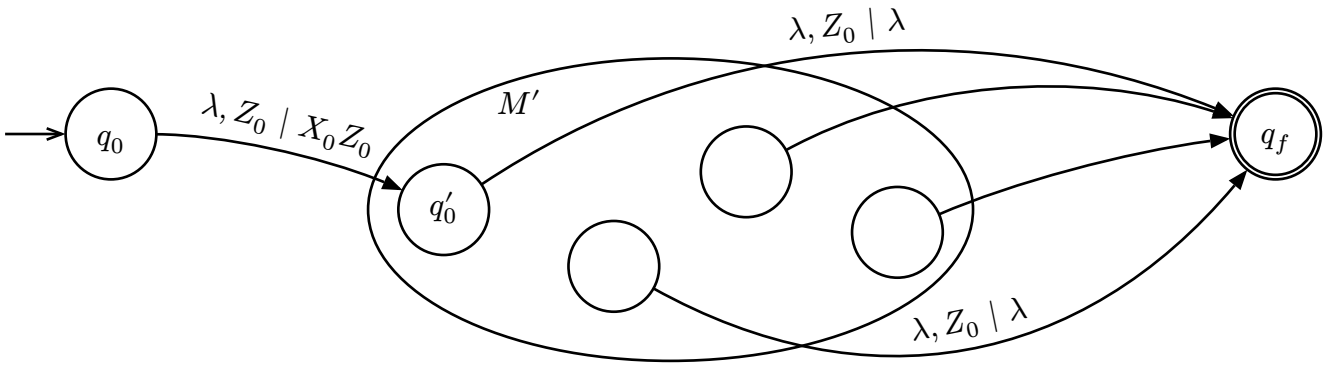
□

Para cada **APND** $M' = \langle Q', \Sigma, \Gamma', \delta', q'_0, X_0, \emptyset \rangle$ existe un **APND** $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$, tal que:

$$\mathcal{L}_\lambda(M') = \mathcal{L}(M)$$

Demostración:

Definimos $M = \langle Q' \cup \{q_0, q_f\}, \Sigma, \Gamma' \cup \{Z_0\}, \delta, q_0, Z_0, \{q_f\} \rangle$, donde



- $\delta(q_0, \lambda, Z_0) = \{(q_0', X_0 Z_0)\} \leftarrow M$ entra al estado inicial de M' con $X_0 Z_0$ en la pila, así evitamos vaciar la pila.
- $\forall q \in Q', \forall x \in \Sigma\{\lambda\}, \forall Z \in \Gamma', \delta(q, x, Z) = \delta'(q, x, Z) \leftarrow M$ mantiene las transiciones de M'.
- $\forall q \in Q', (q_f, \lambda) \in \delta(q, \lambda, Z) \leftarrow$ Cuando se vacía la pila simulada en M', M salta al estado final q_f.

Nos queda por argumentar que $\omega \in \mathcal{L}_\lambda(M')$ si y solo si $\omega \in \mathcal{L}(M)$.

- Si $\omega \in \mathcal{L}_\lambda(M')$ entonces $(q_0', \omega, X_0) \overset{*}{\underset{M'}{\vdash}} (q, \lambda, \lambda)$.

Con lo que al tener encuenta la definición de M asegura

$$(q_0, \omega, Z_0) \underset{M}{\vdash} (q_0', \omega, X_0 Z_0) \overset{*}{\underset{M}{\vdash}} (q, \lambda, Z_0) \underset{M}{\vdash} (q_f, \lambda, \lambda),$$
$$\text{y por lo tanto } \omega \in \mathcal{L}(M).$$

- Si $\omega \in \mathcal{L}(M)$ entonces

$$(q_0, \omega, Z_0) \underset{M}{\vdash} (q_0', \omega, X_0 Z_0) \overset{*}{\underset{M}{\vdash}} (q, \lambda, Z_0) \underset{M}{\vdash} (q_f, \lambda, \lambda),$$
$$\text{pero por definición de } M,$$
$$(q_f, \lambda, X_0 Z_0) \overset{*}{\underset{M}{\vdash}} (q, \lambda, Z_0) \text{ si y solo si } (q_0', \omega, X_0) \overset{*}{\underset{M'}{\vdash}} (q, \lambda, \lambda).$$
$$\text{Concluimos } (q_0', \omega, X_0) \overset{*}{\underset{M'}{\vdash}} (q, \lambda, \lambda), \text{ y por lo tanto } \omega \in \mathcal{L}_\lambda(M').$$

□

5.1.7. Los autómatas de pila determinísticos por estado final tienen mayor poder expresivo que los autómatas de pila determinísticos por pila vacía

Los lenguajes que aceptan los **autómatas de pila determinísticos por pila vacía**, son exclusivamente **libres de prefijos**.
Es decir, sea L en un lenguaje aceptado por un **APD** por pila vacía, entonces

$$\text{si } \alpha \in L \Longrightarrow \forall \beta \neq \lambda, \alpha\beta \in L.$$

Por lo que el poder expresivo de los **autómatas de pila determinísticos por estado final** es **mayor** que el de los **autómatas de pila determinísticos por pila vacía**, pues estos últimos no pueden reconocer lenguajes que no sean libres de prefijos.

5.1.8. Preguntas

1. Si $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset \rangle$ es un autómata de pila entonces cada palabra $\omega \in L_\lambda(M)$ es reconocida por M en a lo sumo $|\omega| * \#Q * \#\Gamma$ transiciones.
   Es decir, $\exists n \leq |\omega| * \#Q * \#\Gamma, \exists p \in Q$, tal que $(q_0, \omega, Z_0) \overset{n}{\vdash} (q_m, \lambda, \lambda)$.

Esta afirmación es **verdadera**.

Demostración:

Sabemos que $|\omega| * \#Q * \#\Gamma$ es el total de «estados» posibles para hacer una transición que puede tener el autómata M al procesar la cadena ω.

Luego, asumiendo que $\omega \in L_\lambda(M)$, entonces existe una secuencia de configuraciones $(q_0, \omega, Z_0) \overset{*}{\vdash} (q_m, \lambda, \lambda)$, con $q_m \in Q$.

Supongamos que no existe $n \leq |\omega| * \#Q * \#\Gamma$ tal que $(q_0, \omega, Z_0) \overset{n}{\vdash} (q_m, \lambda, \lambda)$.
Es decir, $(q_0, \omega, Z_0) \overset{n}{\vdash} (q_m, \lambda, \lambda)$ con $n > |\omega| * \#Q * \#\Gamma$.

Si extendemos la secuencia de configuraciones tenemos algo del estilo:

$$(q_0, \omega_0, Z_0) \vdash (q_1, \omega_1, \gamma_1) \vdash ... \vdash (q_{m-1}, \omega_{m-1}, \gamma_{m-1}) \vdash (q_m, \lambda, \lambda).$$

Sin embargo, dijimos que $n > |\omega| * \#Q * \#\Gamma$, lo que implica que n es mayor al total de configuraciones posibles.
Es decir, tiene que haber al menos una configuración repetida en la secuencia.

$$(q_0, \omega_0, Z_0) \vdash ... \vdash \underbrace{(q_i, \omega_i, Z_i\gamma_j) \vdash ... \vdash (q_i, \omega_i, Z_i\gamma_k)}_{\text{ciclo}} \vdash ... \vdash (q_{m-1}, \omega_{m-1}, \gamma_{m-1}) \vdash (q_m, \lambda, \lambda).$$

Por lo que podríamos quitar ese ciclo.
Si al quitarlo nos queda una secuencia de configuraciones $(q_0, \omega, Z_0) \overset{n}{\vdash} (q_m, \lambda, \lambda)$, con $n > |\omega| * \#Q * \#\Gamma$, resta repetir el argumento, hasta que no quede ningún ciclo.

Luego no va a quedar $(q_0, \omega, Z_0) \overset{n}{\vdash} (q_m, \lambda, \lambda)$ con $n \leq |\omega| * \#Q * \#\Gamma$, lo que contradice la suposición de que no existía tal n.

□

2. Dado un autómata finito $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, dar un autómata de pila $M' = \langle Q', \Sigma, \Gamma', \delta', q_{0'}, Z'_0, \emptyset \rangle$ tal que $\mathcal{L}(M) = \mathcal{L}_\lambda(M')$.

Este enunciado **se puede hacer**.

Justificación:

Te hacés el APND por estado final $M'' = \langle Q, \Sigma, \{Z_0\}, \delta'', q_0, Z_0, F \rangle$, donde replicás el comportamiento (δ) del AF M en M'' ignorando la pila.
Luego, pasamos de M'' a M', donde M' es el autómata de pila por pila vacía que acepta el lenguaje de M''.

3. Consideremos la demostración del teorema que afirma que para cada autómata $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ existe un autómata M' tal que $\mathcal{L}(M) = \mathcal{L}_\lambda(M')$.
   ¿Si M es determinístico, entonces el autómata M' construido en la demostración también lo es?

Esta pregunta es **falsa**.

Justificación:

Los estados finales tienen transiciones λ que te llevan a vaciar la pila. Capaz no tenías que tomar esa transición, pues ese estado final era en realidad un estado intermedio para la aceptación de la cadena.
Respondiendo directamente a la pregunta, no pues los estados finales pueden tener transiciones que llevan a otro estado (distinto de las λ), pues pueden ser estados intermedios.

4. Consideremos la demostración del teorema que afirma que dado $M' = \langle Q', \Sigma, \Gamma', \delta', q_{0'}, X_0, \emptyset \rangle$ existe un autómata M tal que $\mathcal{L}_\lambda(M') = \mathcal{L}(M)$.
   ¿Si M' es determinístico, entonces el autómata M construido en la demostración también lo es?

Esta pregunta es **verdadera**.

Justificación:

Si podés tomar la transición λ con Z_0 en el tope de la pila para ir al estado final, es porque la vaciaste en M'. Recordar que no hay transiciones que tomen Z_0 como tope de la pila en M'. Asumiendo que M' es determinístico, entonces M también lo es.

5.2. Gramáticas libres de contexto

Una gramática $G = \langle V_N, V_T, P, S \rangle$ es libre de contexto si las producciones en P son de la forma:

$$A \to \alpha, \text{ con } A \in V_N \text{ y } \alpha \in (V_N \cup V_T)^*.$$

5.2.1. Derivación de la relación $\Rightarrow$

Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática.
Si $\alpha, \beta, \gamma_1, \gamma_2 \in (V_N \cup V_T)^*$ y $\alpha \to \beta \in P$, entonces

$$\gamma_1 \alpha \gamma_2 \Rightarrow \gamma_1 \beta \gamma_2.$$

La relación $\Rightarrow$ es un subconjunto de $(V_N \cup V_T) \times (V_N \cup V_T)$ y significa derivar en un solo paso.
Las relaciones $\overset{*}{\Rightarrow}$ y $\overset{+}{\Rightarrow}$ son la clausura transitiva y reflexo-transitiva respectivamente (uno o más pasos de derivación, y cero o más pasos).

Si $\alpha \in (V_N \cup V_T)^*$ y $S \overset{*}{\Rightarrow} \alpha$, entonces decimos que α es una **forma sentencial** de G.

5.2.2. Lenguaje generado por una gramática G

Dada una gramática $G = \langle V_N, V_T, P, S \rangle$, el lenguaje generado por G es:

$$\mathcal{L}(G) = \left\{ \omega \in V_T^* : S \overset{*}{\Rightarrow} \omega \right\}.$$

5.3. Relación entre autómatas de pila y gramáticas libres de contexto

5.3.1. Todo lenguaje generado por una gramática libre de contexto, tiene un autómata de pila que lo acepta

Para cada gramática G libre de contexto existe un autómata de pila M tal que

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

5.3.1.1. Demostración

Sea $G = \langle V_N, V_T, P, S\rangle$ una gramática libre de contexto, definimos el **APND**

$$M = \langle \{q\}, V_T, V_N \cup V_T, \delta, q, S, \emptyset\rangle$$

donde $\delta : Q \times (V_T \cup \{\lambda\}) \times (V_N \cup V_T) \to \mathcal{P}(Q \times (V_N \cup V_T)^*)$ es tal que:

- $(A \to \alpha) \in P \iff (q, \alpha) \in \delta(q, \lambda, A)$.
- $\forall x \in V_T, \delta(q, x, x) = \{(q, \lambda)\}$.

Queremos ver que

$$\forall \omega \in V_T^*, S \overset{+}{\Rightarrow} \omega \text{ si y solo si } (q, \omega, S) \overset{+}{\vdash} (q, \lambda, \lambda).$$

Para ello, vamos primero a demostrar un lema.

Lema:

$$\forall A \in V_N, \forall \omega \in V_T^*, A \overset{+}{\Rightarrow} \omega \text{ si y solo si } (q, \omega, A) \overset{+}{\vdash} (q, \lambda, \lambda).$$

Demostración del lema:

Por inducción en m, la cantidad de pasos de la derivación.

- **Caso base:** $m = 1$.
  Queremos ver que $A \overset{1}{\Rightarrow} \omega$ si y solo si $(q, \omega, A) \overset{1}{\vdash} (q, \lambda, \lambda)$.
  Tenemos $A \overset{1}{\Rightarrow} \omega$ para $\omega = x_1 ... x_k$ con $k \geq 0$.
  Esto es lo mismo que decir que $(A \to x_1 ... x_k) \in P$.
  Y esto vale sii $(q, x_1 ... x_k) \in \delta(q, \lambda, A)$, por definición del autómata M.
  Por lo tanto, $(q, x_1 ... x_k, \underset{k}{A}) \overset{1}{\vdash}$. Luego, nuevamente por definición del autómata M, tenemos que $(q, x_1 ... x_k, x_1 ... x_k) \vdash (q, \lambda, \lambda)$.
- **Paso inductivo:**
  Asumamos que vale para todo $j < m$.
  Es decir, $\forall A \in V_N, \forall \omega \in V_T^*, \forall j < m, A \overset{j}{\Rightarrow} \omega$ si y solo si $(q, \omega, A) \overset{j}{\vdash} (q, \lambda, \lambda)$.
  Por definición de la derivación, $A \overset{m}{\Rightarrow} \omega$ si y solo si $(A \to X_1 ... X_k) \in P$, tal que $\forall i \in \mathbb{N}_{\leq k}, X_i \overset{m_i}{\Rightarrow} \omega_i$, para algún $m_i < m \wedge \omega = \omega_1 ... \omega_k$.
  Por definición de M, $(A \to X_1 ... X_k) \in P \iff (q, X_1 ... X_k) \in \delta(q, \lambda, A)$.
  Dicho de otra forma, $(A \to X_1 ... X_k) \in P \iff (q, \omega, A) \vdash (q, \omega, X_1 ... X_k)$.
  - Si $X_i \in V_N$, entonces, por **hipótesis inductiva**, $(q, \omega_i, X_i) \overset{+}{\vdash} (q, \lambda, \lambda)$.
  - Si $X_i \in V_T$, entonces $X_i = \omega_i$ y por definición de M, $(q, \omega_i, X_i) \overset{1}{\vdash} (q, \lambda, \lambda)$.

  Por lo tanto,

$$(q, \omega, A) \vdash (q, \omega_1 ... \omega_k, X_1 ... X_k) \overset{+}{\vdash} (q, \omega_2 ... \omega_k, X_2 ... X_k) \overset{+}{\vdash} (q, \omega_k, X_k) \overset{+}{\vdash} (q, \lambda, \lambda).$$

□

Volviendo a la demostración del teorema,
El lema de recién dice que

$$\forall A \in V_N, \forall \omega \in V_T^*, A \overset{+}{\Rightarrow} \omega \text{ si y solo si } (q, \omega, A) \overset{+}{\vdash} (q, \lambda, \lambda).$$

Tomando $A = S$, tenemos que

$$\forall \omega \in V_T^*, S \overset{+}{\Rightarrow} \omega \text{ si y solo si } (q, \omega, S) \overset{+}{\vdash} (q, \lambda, \lambda).$$

Que es lo que queríamos probar.

□

5.3.2. Todo lenguaje aceptado por un autómata de pila, es libre de contexto

Si M es un autómata de pila, entonces $\mathcal{L}_\lambda(M)$ es libre de contexto.

5.3.2.1. Demostración

Dado el autómata de pila $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$, definimos $G = \langle V_N, V_T, P, S \rangle$, donde

- S es un símbolo nuevo.
- $V_N = \{[q, Z, p] : q \in Q, Z \in \Gamma, p \in Q\} \cup \{S\}$.
- $V_T = \Sigma$.
- Con la siguientes reglas para las producciones:
  - $\forall q \in Q, (S \to [q_0, Z_0, q]) \in P$.
  - $([q, Z, q_1] \to x) \in P \iff (q_1, \lambda) \in \delta(q, x, Z)$.
  - $([q, Z, q_1] \to \lambda) \in P \iff (q_1, \lambda) \in \delta(q, \lambda, Z)$.
  - Para cada $q, q_1, q_2, ..., q_{m+1} \in Q, x \in \Sigma, Z, Y_1, ..., Y_m \in \Gamma$:
    - $([q, Z, q_{m+1}] \to x[q_1, Y_1, q_2]...[q_m, Y_m, q_{m+1}]) \in P \iff (q_1, Y_1...Y_m) \in \delta(q, x, Z)$.
    - $([q, Z, q_{m+1}] \to [q_1, Y_1, q_2]...[q_m, Y_m, q_{m+1}]Y) \in P \iff (q_1, Y_1...Y_m) \in \delta(q, \lambda, Z)$.

G es tal que su derivación más a la izquierda es una simulación de M.

Queremos ver que $\omega \in \mathcal{L}_\lambda(M) \iff \omega \in \mathcal{L}(G)$.

Para ello vamos a demostrar primero un lema.
Lema:

$$\forall q \in Q, \forall Z \in \Gamma, \forall p \in P, (q, \omega, Z) \underset{M}{\overset{+}{\vdash}} (p, \lambda, \lambda) \iff [q, Z, p] \underset{G}{\overset{+}{\Rightarrow}} \omega.$$

Demostración del lema:

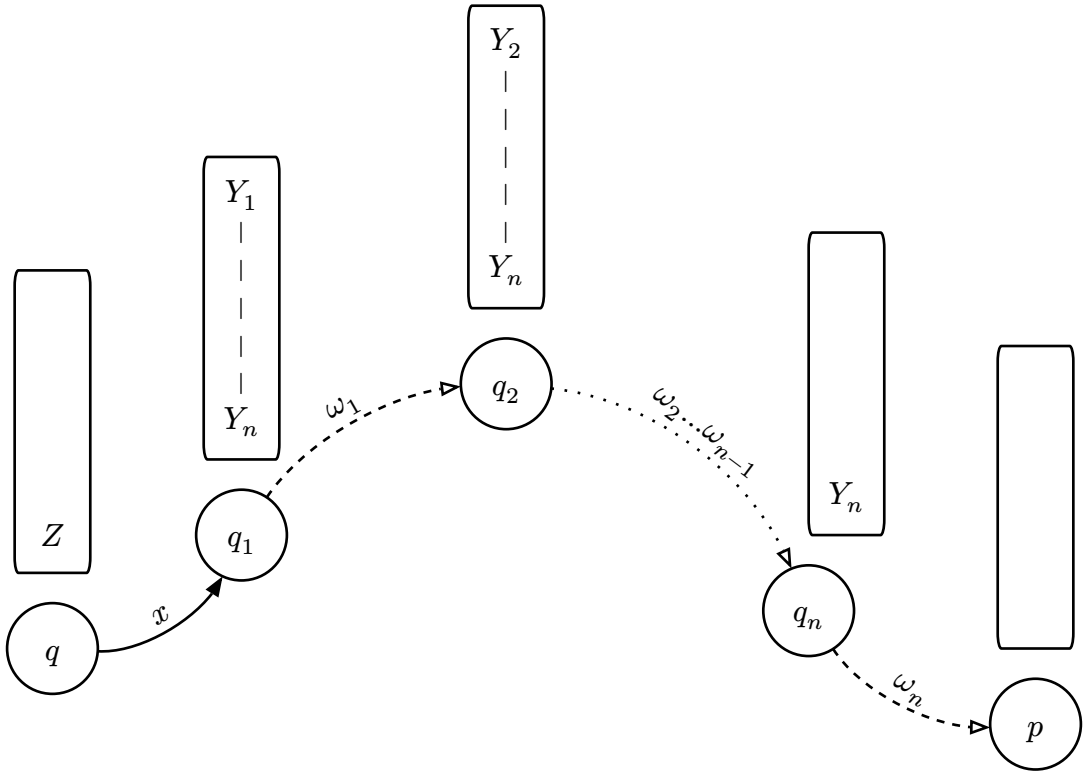
1. $\Rightarrow$) Veamos por inducción que, $\forall i \in \mathbb{N}_{\geq 1}$, se cumple que

$$\forall q \in Q, \forall Z \in \Gamma, \forall p \in P, (q, \omega, Z) \underset{M}{\overset{i}{\vdash}} (p, \lambda, \lambda) \Longrightarrow [q, Z, p] \underset{G}{\overset{i}{\Rightarrow}} \omega.$$

Sea $x \in \Sigma \cup \{\lambda\}$.

- **Caso base:** $i = 1$.
  Tenemos $(q, \omega, Z) \underset{M}{\overset{1}{\vdash}} (p, \lambda, \lambda)$, lo que implica que $(p, \lambda) \in \delta(q, x, Z)$.
  Luego, por definición de G, $([q, Z, p] \to x) \in P$, o, lo que es lo mismo, $[q, Z, p] \underset{G}{\overset{1}{\Rightarrow}} x$.
- **Paso inductivo:** $i > 1$.
  Tenemos $\omega = x\omega'$ con $\omega' \in \Sigma^*$ y $(q, \omega, Z) \underset{M}{\overset{i}{\vdash}} (p, \lambda, \lambda)$.
  Existen $Y_1, ..., Y_n \in \Gamma$ tales que $(q, x\omega', Z) \underset{M}{\vdash} (q_1, \omega', Y_1, ..., Y_n) \underset{M}{\overset{i-1}{\vdash}} (p, \lambda, \lambda)$.

Necesariamente ω' se descompone como $\omega' = \omega'_1 ... \omega'_n$, tales que para $1 \leq j \leq n, \omega_1 ... \omega_j$ hacen que Y_j quede en el tope de la pila.



Existen $q_2, ..., q_{n+1} \in Q$ tales que $1 \leq j \leq n, k_i < i, \left(q_j, \omega'_j, Y_j\right) \overset{k_i}{\underset{M}{\vdash}} \left(q_{j+1}, \lambda, \lambda\right)$.

Por **hipótesis inductiva**,

para cada $1 \leq j \leq n$, si $\left(q_j, \omega'_j, Y_j\right) \overset{k_i}{\underset{M}{\vdash}} \left(q_{j+1}, \lambda, \lambda\right)$, entonces $\left[q_j, Y_j, q_{j+1}\right] \overset{*}{\underset{G}{\Rightarrow}} \omega'_j$.

Pero en G tenemos la producción $\left([q, Z, q_{n+1}] \to x[q_1, Y_1, q_2]...[q_n, Y_n, q_{n+1}]\right) \in P$.

Usando que para cada j, $\left[q_j, Y_j, q_{j+1}\right] \overset{*}{\underset{G}{\Rightarrow}} \omega'_j$, tenemos que
$[q, Z, q_{n+1}] \overset{*}{\underset{G}{\Rightarrow}} x\omega'_1 ... \omega'_n = x\omega' = \omega$.

2. $\Leftarrow$) Veamos por inducción que, $\forall i \in \mathbb{N}_{\geq 1}$, se cumple que

$$\forall q \in Q, \forall Z \in \Gamma, \forall p \in P, [q, Z, p] \overset{i}{\underset{G}{\Rightarrow}} \omega \Longrightarrow (q, \omega, Z) \overset{i}{\underset{M}{\vdash}} (p, \lambda, \lambda).$$

Sea $x \in \Sigma \cup \{\lambda\}$.

- **Caso base:** $i = 1$.
  Tenemos $[q, Z, p] \overset{1}{\underset{G}{\Rightarrow}} x$, lo que implica que $([q, Z, p] \to x) \in P$.
  Luego, por definición de M, $(p, \lambda) \in \delta(q, x, Z)$, lo que implica que $(q, x, Z) \underset{M}{\vdash} (p, \lambda, \lambda)$.
- **Paso inductivo:** $i > 1$.
  Si $[q, Z, p] \overset{i}{\underset{G}{\Rightarrow}} \omega$, entonces $[q, Z, p] \underset{G}{\Rightarrow} x[q_1, Y_1, q_2]...[q_n, Y_n, p] \overset{i-1}{\underset{G}{\Rightarrow}} \omega$.

  Descomponemos ω como $\omega = x\omega_1 ... \omega_n$ tal que para cada $1 \leq j \leq n$, cada derivación toma menos de i pasos: $\left[q_j, Y_j, q_{j+1}\right] \overset{k_i}{\underset{G}{\Rightarrow}} \omega_j$, con $k_i < i$.

  Por **hipótesis inductiva**, para cada $1 \leq j \leq n, \left(q_j, \omega_j, Y_j\right) \overset{*}{\underset{M}{\vdash}} \left(q_{j+1}, \lambda, \lambda\right)$.

Entonces, $(q_j, \omega_j, Y_j, Y_{j+1}, ..., Y_n) \overset{*}{\underset{M}{\vdash}} (q_{j+1}, \lambda, Y_{j+1}, ..., Y_n)$.
Partimos de $[q, Z, p] \underset{G}{\Rightarrow} x[q_1, Y_1, q_2]...[q_n, Y_n, p]$.
Por definición de M, $(q, x, Z) \underset{M}{\vdash} (q_1, \lambda, Y_1...Y_n)$.
Llamando p al q_{n+1}, obtenemos $(q, xy_1...y_n, Z) \underset{M}{\vdash} (q_1, y_1...y_n, Y_1...Y_n) \overset{*}{\underset{M}{\vdash}} (p, \lambda, \lambda)$.

□

Volviendo a la demostración del teorema,
El lema de recién dice que

$$\forall q \in Q, \forall Z \in \Gamma, \forall p \in P, (q, \omega, Z) \overset{+}{\underset{M}{\vdash}} (p, \lambda, \lambda) \iff [q, Z, p] \overset{+}{\underset{G}{\Rightarrow}} \omega.$$

Tomando $q = q_0$ y $Z = Z_0$, tenemos que

$$\forall p \in P, (q_0, \omega, Z_0) \overset{+}{\underset{M}{\vdash}} (p, \lambda, \lambda) \iff [q_0, Z_0, p] \overset{+}{\underset{G}{\Rightarrow}} \omega.$$

Por la definición de G, $(S \to [q_0, Z_0, q]) \in P$, entonces,

$$\forall p \in P, (q_0, \omega, Z_0) \overset{+}{\underset{M}{\vdash}} (p, \lambda, \lambda) \iff S \overset{+}{\underset{G}{\Rightarrow}} \omega.$$

O, lo que es lo mismo,

$$\omega \in \mathcal{L}_\lambda(M) \iff \omega \in \mathcal{L}(G).$$

□

6. Lenguajes libres de contexto: lema de pumping, algoritmos de decisión, propiedades de clausura, gramáticas ambiguas y lenguajes inherentemente ambiguos

6.1. Lenguajes libres de contexto

Los lenguajes libres de contexto son:

- Exactamente los lenguajes generados por las gramáticas libres de contexto $G = \langle V_N, V_T, P, S \rangle$.
- Exactamente los lenguajes aceptados por los autómatas de pila $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$.

Y, como veremos más adelante, se reconocen en tiempo cúbico, tomando como parámetro la longitud de la palabra *(algoritmo CYK)*.

6.2. Lema de pumping para lenguajes libres de contexto

Para todo lenguaje L libre de contexto, $\exists n > 0$ tal que para $\forall \alpha \in L$ con $|\alpha| \geq n$:

- Existe una descomposición de α en cadenas r, x, y, z, s, es decir, $\alpha = rxyzs$.
  - Cumpliendo que:
    - $|xyz| \leq n$.
    - $|xz| \geq 1$.
    - $\forall i \geq 0$, la cadena $rx^iyz^is \in L$.

6.2.1. Demostración

Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto tal que $L = \mathcal{L}(G)$.
Sea $a = \text{máx}\{k : (k = |\beta|, (A \to \beta) \in P, \beta \neq \lambda) \vee (k = 1, (A \to \lambda) \in P)\}$.

Tomemos $n = a^{|V_N|+1}$.
Sea $\alpha \in L$ tal que $|\alpha| \geq n$.
Sea $\mathcal{T}(S)$ un árbol de derivación de la cadena α en G, de altura mínima.
Aún no lo vimos, pero una misma cadena α en una gramática G puede tener $1, 2$, o ∞ árboles de derivación. De entre todos nos quedamos cualquiera que posea altura mínima.

Por el lema visto en la sección 1.5.5.5, $a^h \geq |\alpha|$, con h la altura de $\mathcal{T}(S)$.

Por lo tanto, $a^h \geq |\alpha| \geq n = a^{|V_N|+1}$.

Separemos la demostración en dos casos:

1. $a = 0$.

Por definición de a, $a = 0$ no es posible.
$\to a \geq 1$.

2. $a = 1$.

Si $a = 1$, implica que el lenguaje $\lambda \in L$ o que todas las cadenas $\neq \lambda$ en L tienen longitud 1.
Es decir, en L las cadenas no tienen longitud mayor a 1.
Entonces, seleccionando $n = 2$, el lema de pumping se hace trivialmente verdadero (pues es

cierto que $\exists n$ tal que toda la implicación se hace verdadera; en particular el antecedente $|\alpha| \geq n$ sería falso, y por lo tanto la implicación sería verdadera).

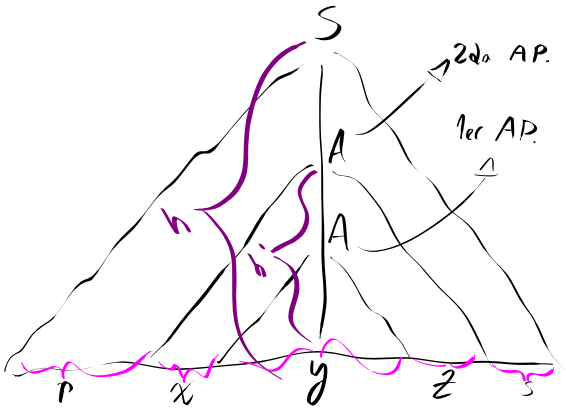
3. $a \geq 2$.

Luego, $h \geq |V_N| + 1$.
Esto $\uparrow$ vale **sii** $a \geq 2$.

Entonces, $h \geq |V_N| + 1$ quiere decir que la altura del árbol de derivación de la cadena α va a ser mayor o igual a la cantidad de no terminales de la gramática G, más uno.
Es decir, va a existir algún símbolo en α tal que su camino de longitud será $h \geq |V_N| + 1$.

Luego, como la cantidad de símbolos no terminales es $|V_N|$, entonces en ese camino de longitud h va a haber al menos un no terminal que se repite.

Llamemos A a ese no terminal repetido. Recorriendo dicho camino **en forma ascendente**, consideremos la primera y la segunda aparición de dicho no terminal, tal como se ve en la figura.



La segunda aparición de A da lugar a la cadena xyz.
Como tenemos la garantía de que esa segunda aparición de A está
a una distancia $h' \leq |V_N| + 1$ respecto de la base (no hay no termnales repetidos en el subárbol $\mathcal{T}(A)$ de la cadena y de altura h', pues agarramos la primera aparición), entonces es cierto que

$$|xyz| \underset{\text{lema de la sección 1.5.5.5}}{\leq} h' \leq |V_N| + 1 = n.$$

Notar que x y z no pueden ser simultáneamente nulas, ya que sino el árbol de derivación para α no sería de altura mínima.

Es decir, como el árbol considerado en esta demostración es de **altura mínima** de entre todos aquellos que sirven para generar la cadena α, siempre se puede encontrar un camino de longitud máxima entre la raíz (S) y alguna hoja, en el que x y z no sean simultáneamente nulas.

Esto ocurre porque, de no ser así, todos los caminos que van desde la raíz S hasta una hoja, que además sean de longitud máxima (o sea, igual a la altura h del árbol) podrían ser «colapsados» y entonces, de este modo, podría obtenerse un árbol de derivación para la cadena α que tendría **menor altura**, lo que es una **contradicción**.

Luego, existen cadenas $r, x, y, z, s \in V_T^*$ tales que $\alpha = rxyzs$, $|xyz| \leq n$, $|xz| \geq 1$ y

$$S \overset{*}{\Rightarrow} rAs \overset{*}{\Rightarrow} rxAzs \overset{*}{\Rightarrow} rxyzs.$$

Por lo tanto tenemos $A \overset{*}{\Rightarrow} y$ y también $A \overset{*}{\Rightarrow} xAz$.

Demostremos ahora que $\forall i \geq 0$, $rx^i yz^i s \in L$, por inducción en i.

- **Caso base:** $i = 0$.
  Tenemos que $S \overset{*}{\Rightarrow} rAs \overset{*}{\Rightarrow} rys$.
  Por lo tanto, $rx^0 yz^0 s = rys \in L$.
- **Paso inductivo:** $i > 0$.
  Asumimos que $rx^i yz^i s \in L$.
  Veamos que se cumple para $i + 1$. Sabemos que $S \overset{*}{\Rightarrow} rx^i Az^i s \overset{*}{\Rightarrow} rx^i yz^i s$.

  Pero como $A \overset{*}{\Rightarrow} xAz$ podemos decir que $rx^i Az^i s \overset{*}{\Rightarrow} rx^{i+1} yz^{i+1} s$.
  Luego nos queda $S \overset{*}{\Rightarrow} rx^i Az^i s \overset{*}{\Rightarrow} rx^{i+1} Az^{i+1} s \overset{*}{\Rightarrow} rx^{i+1} yz^{i+1} s$.

  Por lo tanto, $rx^{i+1} yz^{i+1} s \in L$.

□

6.3. Algoritmos de decisión para lenguajes libres de contexto

6.3.1. Decidir si un lenguaje libre de contexto es vacío

Sea n la constante del lema de pumping para lenguajes libres de contexto.
$L = \emptyset \iff$ ninguna palabra de longitud menor que n pertenece a L.

Si hubiera una de longitud n o más, por el lema de pumping también habría una de longitud menor que n.

```
1:  function ObtenerNoTerminalesActivos(G = ⟨V_N, V_T, P, S⟩)
2:      ▷ Algoritmo para obtener el conjunto de no terminales activos de una gramática.
3:
4:      i ← 1
5:      N_0 ← ∅
6:      N_1 ← {A : (A → α) ∈ P, α ∈ V_T^*}
7:      while N_i ≠ N_{i−1} do
8:          i ← i + 1
9:          N_i ← {A : (A → α) ∈ P, α ∈ (N_{i−1} ∪ V_T)^*} ∪ N_{i−1}
10:     return N_i
```

```
1:  function ElLenguajeGeneradoPorLaGramáticaEsVacío?(G = ⟨V_N, V_T, P, S⟩)
2:      ▷ Algoritmo para decidir si el lenguaje generado por una gramática es vacío.
3:
4:      V_{N_activos} ← ObtenerNoTerminalesActivos(G)
5:      return S ∉ V_{N_activos}
```

6.3.2. Decidir si un lenguaje libre de contexto es finito

Sea n la constante del lema de pumping para lenguajes libres de contexto.
L es finito $\iff$ ninguna palabra de longitud entre n y $2n - 1$ pertenece a L.

6.3.2.1. Demostración

- $\Rightarrow$)

Asumimos que L es finito.
Supongamos que tiene una palabra α de longitud entre n y $2n-1$.
Por el lema de pumping, esa palabra se puede descomponer como $\alpha = rxyzs$ con $|xyz| \leq n$ y $|xz| \geq 1$, haciendo cierto que $rx^iyz^is \in L$ para todo $i \geq 0$.
Sin embargo, eso implica que L tiene infinitas palabras de longitud mayor que n. Absurdo.

- $\Leftarrow$)

Asumimos que L no tiene ninguna palabra de longitud entre n y $2n-1$.
Supongamos que L es infinito.
Notar que si todas las palabras de L tienen longitud $< n$, entonces L es finito trivialmente.
Sea α la palabra en L de longitud más corta, tal que su longitud sea mayor o igual que $2n$.
El lema de pumping afirma que existe una palabra más corta que α tal que esa palabra también pertenece L. Luego, hay una factorización $\alpha = rxyzs$ con $|xyz| \leq n$, $|xz| \geq 1$ y tenemos que $rx^0yz^0s = rys \in L$.

Dado que $|xyz| \leq n$ y $|xz| \geq 1$,

$$|\alpha| - n \leq |\alpha| - |xyz| = |rs| \leq |rys| = |\alpha| - |xz| \leq |\alpha| - 1.$$

Si $|\alpha| = 2n$, nos queda $n \leq |rys| \leq 2n-1$, que contradice nuestro antecedente.
Si $|\alpha| > 2n$, entonces $|rys| < |\alpha|$. Luego, o bien $|rys|$ es entre n y $2n-1$ y llegamos a una contradicción, o bien $|rys| \geq 2n$, contradiciendo que α era la palabra de longitud más corta, tal que su longitud era $\geq 2n$.

□

6.3.3. Decidir si una palabra pertenece a un lenguaje libre de contexto

Hay un algoritmo para decidir la pertenecia de una palabra a un lenguaje libre de contexto. Se llama algoritmo de parsing CYK o Earley.

6.3.4. Cota en la cantidad de pasos en una derivación

Por el lema visto en la sección 1.7.2, podemos obtener un **corolario** tomando $w = \lambda$.
Sea G libre de contexto y no recursiva a izquierda. Existe una constante c tal que para todo par de no terminales A, B, si $A \overset{i}{\underset{L}{\Rightarrow}} B\alpha$ entonces $i \leq c^2$.

6.4. Propiedades de clausura de los lenguajes libres de contexto

Los lenguajes libres de contexto **están cerrados** por
- unión,
- concatenación,
- reversa,
- y clausura de Kleene.

No están cerrados por intersección, diferencia, ni complemento.

Sin embargo, **la intersección de libre de contexto con regular es libre de contexto**.

6.4.1. Unión ✓

Supongamos $G_1 = \langle V_{N_1}, V_{T_1}, P_1, S_1 \rangle$ y $G_2 = \langle V_{N_2}, V_{T_2}, P_2, S_2 \rangle$ dos gramáticas libres de contexto.
Definimos $G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P, S \rangle$, donde $P = P_1 \cup P_2 \cup \{S \to S_1 | S_2\}$.

6.4.2. Concatenación ✓

Supongamos $G_1 = \langle V_{N_1}, V_{T_1}, P_1, S_1 \rangle$ y $G_2 = \langle V_{N_2}, V_{T_2}, P_2, S_2 \rangle$ dos gramáticas libres de contexto.
Definimos $G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_{T_1} \cup V_{T_2}, P, S \rangle$, donde $P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$.

6.4.3. Reversa ✓

Supongamos $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto donde $L = \mathcal{L}(G)$.
Definimos $G' = \langle V_N, V_T, P', S \rangle$ tal que $L^r = \mathcal{L}(G')$ así:

- Para cada producción $(X \to \alpha) \in P$, ponemos $(X \to \alpha^r) \in P'$.
- Supongamos que $(S \to uXv) \in P$ y $(X \to \alpha) \in P$.
  Entonces, ponemos $(S \to v^r X u^r) \in P'$ y $(X \to \alpha^r) \in P'$.
  Luego, $S \underset{G}{\Rightarrow} u\alpha v$ y $S \underset{G'}{\Rightarrow} v^r \alpha^r u^r$.

Dado que $v^r \alpha^r u^r = (u\alpha v)^r$, tenemos que $S \underset{G'}{\Rightarrow} (u\alpha v)^r$.

6.4.4. Clausura de Kleene ✓

Supongamos $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto.
Definimos $G' = \langle V_N \cup \{S'\}, V_T, P', S' \rangle$, donde $P' = P \cup \{S' \to SS' | \lambda\}$.

6.4.5. Intersección ✗

Sean $L_1 = \{a^i b^j c^j\}$ y $L_2 = \{a^i b^i c^j\}$ dos lenguajes libres de contexto.
Notemos que $L_1 \cap L_2 = \{a^i b^i c^i\}$ no es libre de contexto.

6.4.6. Diferencia ✗

Si lo fuese, entonces $\Sigma^* - L$ debería ser libre de contexto.

6.4.7. Complemento ✗

Supongamos que el complemento fuera libre de contexto.
Entonces, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ sería ser libre de contexto.

□

6.5. Gramáticas ambiguas

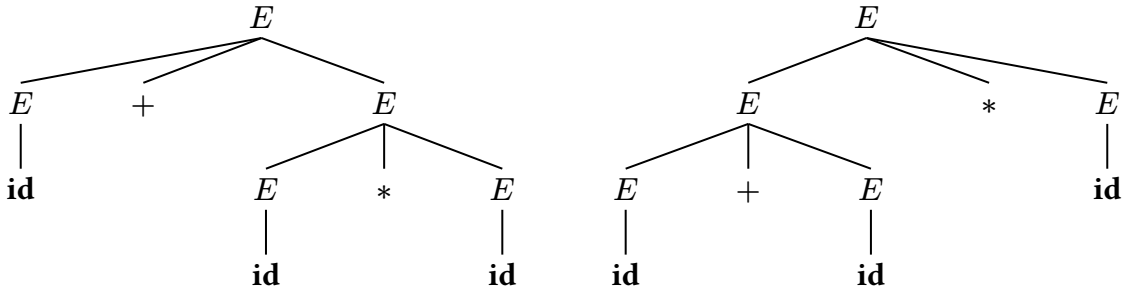
Una gramática libre de contexto G es ambigua si existe una palabra $\alpha \in \mathcal{L}(G)$ con más de una derivación más a la izquierda.

6.5.1. Ejemplo

Dada la gramática $G = \langle \{E\}, \{+, *, \mathbf{id}, \mathbf{const}\}, P, E \rangle$ con

$$(E \to E + E) \in P$$
$$(E \to E * E) \in P$$
$$(E \to \mathbf{id}) \in P$$
$$(E \to \mathbf{const}) \in P$$

Para G podemos dar dos árboles de derivación distintos para la palabra $\mathbf{id} + \mathbf{id} * \mathbf{id}$:



6.6. Lenguaje inherentemente ambiguo

Un lenguaje libre de contexto es inherentemente ambiguo si solamente admite gramáticas ambiguas.

6.6.1. Ejemplo

Este lengueje es inherentemente ambiguo:

$$\{a^n b^m c^m d^n \mid n, m > 0\} \cup \{a^n b^n c^m d^m \mid n, m > 0\}.$$

Es libre de contexto porque es la unión de dos conjuntos que lo son.
Hopcroft y Ullman (1979) mostraron que no hay forma de derivar de manera no ambigua las cadenas de la intersección de ambos lenguajes, es decir del conjunto

$$\{a^n b^n c^n d^n \mid n > 0\},$$

además este conjunto intersección no es libre de contexto.

El problema de si una gramática es ambigua es indecidible.

Es equivalente al Problema de Correspondencia de Post°:
Dadas dos listas de palabras sobre un alfabeto (con al menos dos símbolos) $\alpha_1, ..., \alpha_N$ y $\beta_1, ..., \beta_N$, decidir si hay una secuencia de índices $(i_k)_1^k$ con $K \geq 1$ y $1 \leq i_k \leq N$ tal que $\alpha_{i_1} ... \alpha_{i_K} = \beta_{i_1} ... \beta_{i_K}$.

Por ejemplo,
$\alpha_1 = a, \alpha_2 = ab, \alpha_3 = bba$
$\beta_1 = baa, \beta_2 = aa, \beta_3 = bb.$

Solución: $(3, 2, 3, 1)$ porque

$$\alpha_3\alpha_2\alpha_3\alpha_1 = bba + ab + bba + a = bbaabbaa = bb + aa + bb + baa = \beta_3\beta_2\beta_3\beta_1.$$

6.7. Decisión de lenguajes libres de contexto

Sea G una gramática libre de contexto y $L = \mathcal{L}(G)$.

Hay algoritmos para:
- Decidir si $L = \emptyset$.

- Decidir si L es finito.
- Decidir si L es infinito.
- Decidir si una palabra $\alpha \in \Sigma^*$ pertenece a L, algoritmo CYK.
  En tiempo del orden cúbico en la longitud de w.
  En casos especiales *(determinismo)*, en tiempo lineal.

No hay algoritmos para:

- Decidir si L es regular.
- Decidir si G una gramática libre de contexto es ambigua.
- Decidir si $L_1 = L_2$, con L_1, L_2 lenguajes libres de contexto.
- Decidir si $L = \Sigma^*$.
- Decidir si $L_1 \subseteq L_2$, con L_1, L_2 lenguajes libres de contexto.
- Decidir si $L_1 \cap L_2 = \emptyset$, con L_1, L_2 lenguajes libres de contexto.

6.8. Preguntas

1. Sea L un lenguaje. Si todas las palabras de L validan el lema de pumping para lenguajes libres de contexto, ¿podemos concluir que L es un lenguaje libre contexto?
   No, no vale la $\Leftarrow$.

2. Sea L un lenguaje regular. Demostrar que todas las palabras de L validan el lema de pumping para lenguajes libres de contexto.
   Basta tomar $z = s = \lambda$ en la descomposición de la palabra.

3. Mostrar que $L = \{a^p : p$ es número primo no es libre de contexto.
   Ayuda: asumir L es libre de contexto, y n es la longitud dada por el lema de pumping. Sea m el primer primo mayor o igual que n, considerar $\alpha = a^m$ y bombear $m + 1$ veces.
   Elegís $\alpha = a^m$ tal que m es el primer primo mayor o igual que n, sin perdida de generalidad, podemos decir que la cadena α se puede descomponer como $\alpha = rxyzs$, tal que
   $r = a^b, x = a^c, y = a^d, z = a^e, s = a^f$, donde
   $b, c, d, e, f \geq 0$,
   $f = m - b - c - d - e$,
   $c + d + e \leq n$,
   $c + e \geq 1$ y
   $b + c + d + e + f = m$.

   Luego, tomando $i = m + 1$ nos queda: $\alpha_{i=m+1} = a^{(c+e+1)m}$, que no pertenece a L pues $(c + e + 1)m$ no es primo.

   □

4. Demostrar que un lenguaje regular intersección un lenguaje libre de contexto es libre de contexto.
   Ayuda: definir el autómata de pila que lo reconoce.
   Sea $M_R = \langle Q_R, \Sigma, \delta_R, q_{0_R}, F_R \rangle$ un autómata finito determinístico que reconoce L_R regular y $M_L = \langle Q_L, \Sigma, \Gamma, \delta_L, q_{0_L}, Z_0, F_L \rangle$ un autómata de pila que reconoce L_L libre de contexto.
   Defino $M = \langle Q_R \times Q_L, \Sigma, \Gamma, \delta, \left(q_{0_R}, q_{0_L}\right), Z_0, F_R \times F_L \rangle$, donde δ se define como

$\delta((q_R, q_L), a, Z) = \{((\delta_R(q_R, a), p), \gamma) \mid (p, \gamma) \in \delta_L(q_L, a, Z)\}$.
Basta probar que $x \in \mathcal{L}(M) \iff x \in \mathcal{L}(M_R) \wedge \mathcal{L}(M_L)$, donde $x \in \Sigma^*$.

$$
\begin{aligned}
x \in \mathcal{L}(M) &\iff \delta\Big(\Big(q_{0_R}, q_{0_L}\Big), Z_0, x\Big) \in F \\
&\iff \Big(\Big(\delta_R\Big(q_{0_R}, x\Big), p\Big), \gamma\Big) \in F_R \times F_L, \text{con } (p, \gamma) \in \delta_L\Big(q_{0_L}, x, Z_0\Big) \\
&\iff \delta_R\Big(q_{0_R}, x\Big) \in F_R \wedge (p, \gamma) \in F_L, \text{con } (p, \gamma) \in \delta_L\Big(q_{0_L}, x, Z_0\Big) \\
&\iff \delta_R\Big(q_{0_R}, x\Big) \in F_R \wedge \delta_L\Big(q_{0_L}, x, Z_0\Big) \cap F_L \neq \emptyset \\
&\iff x \in \mathcal{L}(M_R) \wedge x \in \mathcal{L}(M_L).
\end{aligned}
$$

□

5. Demostrar que hay lenguajes que pueden ser reconocidos con un autómata con dos pilas pero no pueden ser reconocidos con un autómata con una sola pila.
   $L = \{a^n b^n c^n \mid n \geq 0\}$ es reconocido por un autómata con dos pilas, pero no por uno con una sola pila.
   ¡También te lo acepta un autómata de cola!, pero, bueno, es porque un autómata de cola es equivalente a una máquina de Turing...

7. Configuraciones ciclantes, lenguajes libres de contexto determinísticos y gramáticas LR

7.1. Configuraciones ciclantes

Es posible que un autómata de pila determinístico realice una cantidad infinita de λ-movimientos desde alguna configuración, es decir, movimientos que no leen de la cinta de entrada.

Definición de configuración ciclante:

Sea $P = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ un autómata de pila determinístico.
Una configuración (q, w, α), con $|\alpha| \geq 1$, cicla si

$$(q, w, \alpha) \vdash (p_1, w, \beta_1) \vdash (p_2, w, \beta_2) \vdash \ldots$$

con $|\beta_i| \geq |\alpha|$ para todo i.

Así, una configuración cicla si P realiza un número infinito de movimientos sin leer ningún símbolo de la entrada y el tamaño de la pila es de tamaño mayor o igual que $|\alpha|$.
La pila puede crecer indefinidamente o ciclar entre distintas cadenas.

7.2. Lenguajes libres de contexto determinísticos

Un lenguaje L es **libre de contexto determinístico** si existe un **autómata de pila determinístico** M tal que $L = \mathcal{L}(M)$.
Se reconocen en tiempo lineal, algoritmo CYK.

Un lenguaje libre de contexto determinístico implica que admiten una gramática no ambigua.

Ejemplos:
- $L = \{a^n b^n\}$
- $L = \{a^i b^i a^j : i, j \geq 1\}$

Contraejemplos:
- $L_0 = \{a^i b^i c^i : i \geq 1\}$ no es libre de contexto (lema de pumping)
- $L_1 = \{a^i b^j a^j : i \neq j\}$ no es libre de contexto (lema de Ogden)
- $L_2 = \{a^i b^j a^j : i \neq j\} \cup \{a^i b^i a^j : i \neq j\}$ no es libre de contexto
- $L_3 = \{a^i b^i a^j : i, j \geq 1\} \cup \{a^i b^j a^j : i, j \geq 1\}$ es libre de contexto no determinístico

7.3. APD sin ciclos

Para todo **autómata de pila determinístico** P **existe otro** autómata de pila determinístico P' tal que $\mathcal{L}(P) = \mathcal{L}(P')$ y P' **no tiene configuraciones ciclantes**.

7.4. Lema

Sea $P = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ un autómata de pila determinístico.
Si $(q, w, A) \overset{n}{\vdash} (q', \lambda, \lambda)$ entonces para toda $A \in \Gamma$ y $\alpha \in \Gamma^*$, $(q, w, A\alpha) \overset{n}{\vdash} (q', \lambda, \alpha)$.

□

7.5. Detección de configuraciones ciclantes

Dado un autómata de pila **determinístico** $P = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$, la función de transición es una función parcial $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to Q \times \Gamma^*$.

La máxima cantidad de transiciones que P puede hacer sin leer de la entrada y sin repetir el estado ni el tope de la pila es $|Q| \times |\Gamma|$.

En cada transición se escriben en la pila a lo sumo ℓ símbolos de Γ.

La cantidad máxima de configuraciones distintas sin leer la entrada contando la pila entera es:

$$|Q| \times (\text{la cantidad de posibles pilas de longitud hasta } |Q|\ell|\Gamma|)$$
$$|Q| \sum_{i=0}^{|Q|\ell|\Gamma|} |\Gamma|^i \underset{\text{progresión geométrica}}{=} |Q| \ \tfrac{|\Gamma|^{|Q|\ell|\Gamma|+1}-1}{|\Gamma|-1}.$$

Este es el máximo de transiciones-λ que P puede hacer sin ciclar.

7.6. APD que leen toda la entrada

Un APD $P = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ es continuo si para toda $w \in \Sigma^*$ existe $p \in Q$ y $\gamma \in \Gamma^*$ tales que $(q_0, w, Z_0) \overset{*}{\vdash} (p, \lambda, \gamma)$.
Es decir, APD P es continuo si lee toda la cadena de entrada.

7.6.1. Lema

Para todo APD, existe otro equivalente y continuo.

7.6.1.1. Demostración del lema por algoritmo

Algoritmo 2.16, Aho Ullman, vol. 1, pág 187.°

Sea $P = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ un APD. Construimos un APD P' equivalente y continuo.

Sean,

$$C_1 = \Big\{(q, Z) \mid (q, \lambda, Z) \text{ es una configuración ciclante y no existe}$$
$$g \in F \text{ para ningún } \alpha \in \Gamma^* \text{ tal que } (q, \lambda, Z) \overset{*}{\vdash} (g, \lambda, \alpha)\Big\},$$

$$C_2 = \Big\{(q, Z) \mid (q, \lambda, Z) \text{ es una configuración ciclante y hay un}$$
$$g \in F \ \text{ y } \ \alpha \in \Gamma^* \text{ tales que } (q, \lambda, Z) \overset{*}{\vdash} (g, \lambda, \alpha)\Big\}.$$

Supongamos P siempre tiene una próxima transición.
Sea $P' = \langle Q \cup \{f, t\}, \Sigma, \Gamma, \delta_0, q_0, Z_0, F \cup \{f\}\rangle$ donde f y t son nuevos.
La función de transición $\delta' : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to Q \times \Gamma^*$ se define así:
- Para todo $(q, Z) \notin (C_1 \cup C_2)$, $\delta'(q, \lambda, Z) = \delta(q, \lambda, Z)$.
- Para todo $(q, Z) \in C_1$, $\delta'(q, \lambda, Z) = (t, Z)$.
- Para todo $(q, Z) \in C_2$, $\delta'(q, \lambda, Z) = (f, Z)$.
- Para todo $a \in \Sigma$ y $Z \in \Gamma$, $\delta'(f, a, Z) = (t, Z)$ y $\delta'(t, a, Z) = (t, Z)$.

□

7.7. Propiedades de clausura lenguajes libres de contexto determinísticos

Están clausurados por:

7.7.1. El complemento de un lenguaje libre de contexto determinístico es otro lenguaje libre de contexto determinístico

Sea $P = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ un APD continuo.
Definimos $P' = \langle Q', \Sigma, \Gamma, \delta', q_0', Z_0, F' \rangle$ un APD donde

$$Q' = \{[q, k] : q \in Q \land k = 0, 1, 2\}$$

El propósito de k es indicar si entre transiciones con consumo de entrada en P pasó o no por un estado final.

$$q_0' = \begin{cases} [q_0,0] \text{ si } q_0 \notin F \\ [q_0,1] \text{ si } q_0 \in F \end{cases}$$
$$F' = \{[q, 2] : q \in Q\}$$

La semántica es la siguiente:
- $[q, 0]$ indica que P no pasó por F.
- $[q, 1]$ indica que P sí pasó por F.
- $[q, 2]$ indica que P no pasó por F y P va a seguir leyendo.

Para todo $q \in Q$, $[q, 2]$ es el estado final al que llega P' antes de que P lea un nuevo símbolo.

Esta parte falta completarla.

7.7.2. Intersección con lenguejes regulares

Es lo mismo que lo hecho en la pregunta 4 de la sección 6.8, solamente faltaría demostrar que si el lenguaje L libre de contexto es determinístico, entonces $L \cap R$ también lo es.

A nivel intuitivo, R regular se lo puede asumir generado por un autómata finito determinístico sin pérdida de generalidad, pues un AFD es equivalente a un AFND.
Luego, con la definición de δ dada en la demostración de la pregunta 4 en la sección 6.8, para cada $\delta((q_R, q_L), a, Z)$ va a existir una única tupla $((p_R, p_L), \gamma)$ tal que $p_R \in F_R$ y $(p_L, \gamma) \in F_L$.

7.7.3. Concatenación de un lenguaje libre de contexto determinístico *(primero)* con un lenguaje regular *(segundo)*

La demostración sale pensando en la concatenación de ambos autómatas.

7.7.4. Prefijos

Si $pre(L)$ es el subconjunto de todas las cadenas de L, las cuales tienen un prefijo que también pertenece a L.
Entonces, el subconjunto de cadenas *(lenguaje)* $pre(L)$ es libre de contexto determinístico.

7.7.5. Mínimo

Si mín(L) es el subconjunto de todas las cadenas de L, las cuales no tienen un prefijo que también pertenece a L.
Entonces, el subconjunto de cadenas *(lenguaje)* mín(L) es libre de contexto determinístico.

7.7.6. Máximo

Si máx(L) es el subconjunto de todas las cadenas de L, las cuales no son prefijo de una cadena más larga que ellas en L.
Entonces, el subconjunto de cadenas *(lenguaje)* máx(L) es libre de contexto determinístico.

No están clausurados por:
- Intersección
- Unión
- Reversa
- Concatenación
- Clausura de Kleene

7.8. Decisión sobre lenguajes libres de contexto determinísticos

Hay algoritmos para:

- ¿$L = \emptyset$?
  Mismo caso que el no determinístico.
- ¿L es finito?
  Pumping.
- ¿L es infinito?
  Pumping.
- ¿L es cofinito?
  Pumping sobre L^c.
- ¿$L = \Sigma^*$?
  Equivalente a preguntarse ¿$L^c = \emptyset$?
- ¿$L_1 = L_2$?
  Sí, por Géraud Sénizergues, premio Gödel 2002.
- ¿$\omega \in L$?
  Sí, en tiempo lineal en la longitud de ω (algoritmo LR).
- ¿L es regular?
  Sí.
  Steans, R. (1967). A Regularity Test for Pushdown Machines. Information and Control, 11, 323-340.
- ¿$L = R$?
- ¿$L \subseteq R$?

No hay algoritmos para:
- ¿$L_1 \subseteq L_2$?
- ¿$L_1 \cap L_2 = \emptyset$?

7.9. Gramáticas LR

Esta parte falta completarla.

7.10. Autómatas de pila enriquecidos (autómata contador)

También llamados autómatas con un contador o autómatas contadores.
Son autómatas de pila, donde el alfabeto de pila tiene exactamente dos símbolos: Z_0 y I.
En cada transición pueden revisar si el contador es Z_0 o no, y pueden incrementarlo o decrementarlo.

Los lenguajes reconocidos por autómatas contadores incluyen a todos los lenguajes regulares pero son un subconjunto propio de los lenguajes reconocidos por autómatas de pila.

- Los autómatas de pila de dos vías (moverse izq./der. en la cadena de entrada α), tienen una pila, control finito y una cinta cuya cabeza puede moverse en ambas direcciones, tienen menos poder computacional que una máquina de Turing.
- Los autómatas con dos o más pilas son equivalentes a una máquina de Turing.
- Los autómatas con tres contadores son equivalentes a una máquina de Turing.
- Los autómatas con dos contadores son equivalentes a una máquina de Turing.
- Los autómatas con una cola son equivalentes a autómatas con dos pilas, por lo tanto, equivalentes a una máquina de Turing.

8. Máquinas de Turing, funciones parcialmente computables y conjuntos computablemente enumerables

8.1. Máquinas de Turing

Una máquina de Turing es una tupla $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, q_f \rangle$ donde
- Q es un conjunto finito de estados.
- Σ es un conjunto finito de símbolos de la cinta, con $B \in \Sigma$.
- $\delta \subseteq Q \times \Sigma \times Q \times (\Sigma \cup \{L, R\})$ es la tabla de instrucciones, finita.
- $q_0 \in Q$ es el estado inicial.
- $q_f \in Q$ es el estado final.

Si no hay restricciones sobre δ decimos que $\mathcal{M}$ es una máquina de Turing no determinística.

Si no hay dos instrucciones en δ que empiezan con las mismas primeras dos coordenadas, decimos que $\mathcal{M}$ es una máquina de Turing determinística.

Por lo tanto, $\delta : Q \times \Sigma \to Q \times (\Sigma \cup \{L, R\})$.

Existe en la literatura un abundante número de definiciones alternativas, pero todas ellas tienen el mismo poder computacional.

La tupla $(q, s, a, q') \in \delta$ se interpreta como:

Si la máquina está en el estado q leyendo en la cinta el símbolo s, entonces escribe/realiza la acción a y pasa al estado q'.

8.1.1. Configuración instantánea de una máquina de Turing

$\mathcal{M}$ está en estado q con la cabeza en el símbolo más a la izquierda de α_2.
Una configuración/descripción instantánea de $\mathcal{M}$ es $\alpha_1 q \alpha_2$, donde $\alpha_1, \alpha_2 \in \Sigma^*$ y $q \in Q$.

Notar que α_1 y α_2 pueden tener blancos adentro.

8.1.2. Transición entre configuraciones instantáneas $\vdash$

Supongamos $X_1 ... X_{i-1} q X_i ... X_n$ es una configuración instantánea de $\mathcal{M}$.

Si $(p, Y) \in \delta(q, X_i)$ con $Y \notin \{L, R\}$, entonces para $i > 1$,

$$X_1 ... X_{i-1} q X_i ... X_n \underset{\mathcal{M}}{\vdash} X_1 ... X_{i-1} p Y ... X_n$$

Si $(p, L) \in \delta(q, X_i)$, entonces para $i > 1$,

$$X_1 ... X_{i-1} q X_i ... X_n \underset{\mathcal{M}}{\vdash} X_1 ... X_{i-2} p X_{i-1} X_i ..., X_n$$

Si $(p, R) \in \delta(q, X_i)$, entonces para $i > 1$,

$$X_1 ... X_{i-1} q X_i ... X_n \underset{\mathcal{M}}{\vdash} X_1 ... X_{i-1} X_i p X_{i+1} ... X_n$$

$\mathcal{M}$ se detiene en una configuración instantánea cuando no hay un próximo movimiento, porque δno tiene una instrucción para eso.

8.1.3. Lenguaje aceptado por una máquina de Turing

El lenguaje aceptado por una máquina de Turing $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, q_f \rangle$ es el conjunto de palabras $w \in (\Sigma \setminus \{B\})^*$ que causan que $\mathcal{M}$ llegue a un estado final,

$$\mathcal{L}(\mathcal{M}) = \left\{ w \in (\Sigma \setminus \{B\})^* : q_0 w \overset{*}{\vdash} \alpha q_f \beta, \text{donde } \alpha, \beta \in \Sigma^* \right\}$$

8.2. Formas de usar una máquina de Turing

1. Aceptadoras de subconjuntos computablemente enumerables en $(\Sigma \setminus \{B\})^*$.
2. Funciones $f : \mathbb{N} \to \mathbb{N}$ Turing computables.
3. Enumeradoras de conjuntos computablemente enumerables en $(\Sigma \setminus \{B\})^*$.

8.3. Lenguajes computablemente enumerables

Un lenguaje $L \subseteq (\Sigma \setminus \{B\})^*$ es computablemente enumerable (abreviado, c.e.) si existe una máquina de Turing $\mathcal{M}$ tal que $\mathcal{L}(\mathcal{M}) = L$.

Equivalentemente,
si $L \subseteq (\Sigma \setminus \{B\})^*$ representa el dominio de $f : \mathbb{N} \to \mathbb{N}$ parcial computable, es decir, si hay una máquina de Turing $\mathcal{M}$ que computa f.

Equivalentemente,
si $L \subseteq (\Sigma \setminus \{B\})^*$ es el lenguaje generado por una máquina de Turing, $\mathcal{G}(\mathcal{M}) = L$.

8.4. Para toda máquina de Turing $\mathcal{M}_1$ multicinta, existe una máquina de Turing $\mathcal{M}_2$ de una sola cinta tal que $\mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2)$

Esta parte falta completarla.

8.5. Para toda máquina de Turing $\mathcal{M}_1$ no determinística, existe una máquina de Turing $\mathcal{M}_2$ determinística tal que $\mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2)$

Esta parte falta completarla.

9. Cheat sheet

9.1. General

- Un AFND es equivalente a un AFD.
- Los APND por pila vacía y por estado final tienen el mismo poder expresivo.
- Un APD por estado final tiene un mayor poder expresivo que un APD por pila vacía.
- Los APD por pila vacía aceptan solamente lenguajes libres de prefijos.
- Todo lenguaje finito es regular.
- Cuando hablamos del n del lema de pumping, podemos decir que:
  - Si es un lenguaje regular, $n =$ cantidad de estados de un AFD que genera L.
  - Si es un lenguaje libre de contexto, $n = a^{|V_N|+1}$ donde,
    - $a =$ la cantidad máxima de hijos que puede tener un nodo.
    - $V_N =$ conjunto de no terminales activos.

9.2. Lenguajes regulares

9.2.1. Hay algoritmos para decidir si:

Sea L un lenguaje regular y $w \in L$ una cadena.

- $w \in L$
  - *Complejidad $O(n)$ si es determinístico, donde n es la longitud de la cadena.*
  - *Si no es determinístico, se puede convertir a determinístico y luego aplicar el algoritmo. El costo de determinizar es exponencial en la cantidad de estado del autómata.*
- $L \neq \emptyset$
  - Basta ver si existe cadena de longitud $< n$ *(n del lema de pumping).*
  - Basta buscar los no terminales activos de la gramática que genera el lenguaje regular, luego ver si S pertenece al conjunto de los no terminales activos ($\in S$).
- L es infinito
  - Basta ver si existe alguna cadena con longitud entre n y $2n-1$ *(n del lema de pumping).*
- $L = \emptyset$
  - Basta ver si **no existe ninguna** cadena de longitud $< n$ *(n del lema de pumping).*
  - Basta buscar los no terminales activos de la gramática que genera el lenguaje regular, luego ver si S **no** pertenece al conjunto de los no terminales activos ($\notin S$).
- L es finito
  - Basta ver si **no existe ninguna** cadena con longitud entre n y $2n-1$ *(n del lema de pumping).*

En la teórica vimos métodos efectivos para pasar de:

- **Autómata finito** a **gramática regular**.
- **Gramática regular** a **autómata finito**.
- **Autómata finito** a **expresión regular**.
- **Expresión regular** a **autómata finito**.

Por lo que, las 3 representaciones de lenguajes regulares son equivalentes.

9.2.2. Están cerrados por:

- Unión

- Intersección
- Complemento
- Concatenación
  - ↑ *Todo esto lo hace un álgebra de Boole.*
- Reversa
- Diferencia → $L_1 - L_2 = L_1 \cap \overline{L_2}$
- Diferencia simétrica → $L_1 \oplus L_2 = (L_1 - L_2) \cup (L_2 - L_1)$
- Clausura de Kleene

9.3. Lenguajes libres de contexto (no determinísticos)

9.3.1. Hay algoritmos para decidir si:

Sea L un lenguaje libre de contexto y $w \in L$ una cadena.

- $w \in L$
  - *Complejidad $O(n^3)$ por el algoritmo CYK, donde n es la longitud de la cadena.*
- Obtener los no terminales activos de la gramática que genera el lenguaje libre de contexto.
  - *Ver algoritmo en el apunte.*
- $L \neq \emptyset$
  - ↑ *Explicación en $*$.*
- L es infinito
  - ↑ *Explicación en $*$.*
- $L = \emptyset$
  - ↑ *Explicación en $*$.*
- L es finito
  - ↑ $*$ *Son los mismos que para lenguajes regulares, solo que el n del lema de pumping es el n lema de pumping para lenguajes libres de contexto.*

En la teórica vimos métodos efectivos para pasar de:

- **Autómata finito** a **gramática regular**.
- **Gramática regular** a **autómata finito**.

9.3.2. No hay algoritmos para decidir si:

Sean L, L_1, L_2 lenguajes libres de contexto

- L es regular
- Una gramática G que genera L es ambigua
- $L = \Sigma^*$
- $L_1 \subseteq L_2$ es libre de contexto
- $L_1 \cap L_2$ es libre de contexto

9.3.3. Están cerrados por:

- Unión
- Intersección con un lenguaje regular
- Concatenación
- Reversa
- Clausura de Kleene

9.3.4. No están cerrados por:

- Intersección
- Complemento
- Diferencia

9.4. Lenguajes libres de contexto (determinísticos)

9.4.1. Hay algoritmos para decidir si:

Sean L, L_1, L_2 lenguajes libres de contexto, R un lenguaje regular y $w \in L$ una cadena.

- $w \in L$
  - *Complejidad $O(n)$ por el algoritmo CYK, donde n es la longitud de la cadena.*
- Obtener los no terminales activos de la gramática que genera el lenguaje libre de contexto.
  - *Ver algoritmo en el apunte.*
- $L \neq \emptyset$
  - $\uparrow$ *Idem no determinísticos.*
- L es infinito
  - $\uparrow$ *Idem no determinísticos.*
- $L = \emptyset$
  - $\uparrow$ *Idem no determinísticos.*
- L es finito
  - $\uparrow$ *Idem no determinísticos.*
- L es cofinito
- $L = \Sigma^*$
- $L_1 = L_2$
- L es regular
- $L = R$
- $L \subseteq R$

9.4.2. No hay algoritmos para decidir si:

Sean L_1, L_2 lenguajes libres de contexto.

- $L_1 \subseteq L_2$
- $L_1 \cap L_2 = \emptyset$

9.4.3. Están cerrados por:

- Complemento
- Intersección con un lenguaje regular
- LR
  - Concatenación de un lenguaje L libre de contexto con un lenguaje regular R.
- $pre(L)$
  - Prefijos de un lenguaje L libre de contexto.
- mín(L)
- máx(L)

9.4.4. No están cerrados por:

- Unión

- Intersección
- Concatenación
- Reversa
- Clausura de Kleene

10. Ejercicios de práctica para el final

10.1. Demostración de que la complejidad de Kolmogorov no es cumputable

Se define la complejidad de Kolmogorov como $K : \Sigma^* \to \mathbb{N}$ tal que

$$K(s) = \text{mín}\{|p| : \Psi^1(p) = s\}.$$

Es decir, $K(s)$ es igual al programa p más chico* que «imprime» -para nosotros, devuelve- s.
* entendido como más chico, cantidad de bits que ocupa el código del programa.

Evidentemente hay una especie de "cota superior" para $K(s)$,

```c
printf("%s", s);
```

es el programa **trivial** más chico que imprime s.

Asumamos que es computable, por lo que puedo usarla en un programa. Basándonos en la tesis de Church, puedo usar un programa en C tranquilamente tal que
↑ Todo para no hacerlo en S++, el rey de la vagancia.

```c
// Es un pseudocódigo, puede contener errores.
int main() {
  const int C = 10000; // Constante C arbitraria, tal que sea mayor a la "longitud" de este programa.
  char *s = "";

  while (K(s) <= C) {
    // Donde next(s) es una función que devuelve la siguiente cadena en el alfabeto Sigma, por ejemplo, en un alfabeto de la "a" a la "z", todas minúsculas.
    *s = next(s);
    // Si *s == "", entonces next(s) == "a", si *s == "a", entonces next(s) == "b", ..., si *s == "z", entonces next(s) == "aa"..
  }

  printf("%s", s);
  return 0;
}
```

Tenemos asumido que $K(s)$ es computable.
Analicemos el comportamiento del while en main.
Este programa va probando todas las cadenas posibles de Σ^*, posiblemente en cierto orden lexicográfico, hasta encontrar una cadena s tal que $K(s) > C$.
Cuando encuentra una cadena s tal que $K(s) > C$, imprime s y termina.
Notar que esa cadena s existe, pues siempre va a existir un string s tal que $K(s) > C$. Pensar en un string «archivo» aleatorio muy, muy, muy, grande.
Sin embargo, la existencia de esa cadena s tal que $K(s) > C$, es un absurdo. Pues el programa main es un programa que imprime s y tiene complejidad de Kolmogorov $\leq C$.
Luego, la complejidad de Kolmogorov no es computable.

□

10.2. Ejercicio 1

Dar un algoritmo que decida si dos expresiones regulares denotan el mismo lenguaje.
Justificar la correctitud.

Sean E_1 y E_2 las dos expresiones regulares.

1. Utilizando el algoritmo visto en la práctica, puedo **construir los autómatas finitos** M_1 y M_2 que reconocen los lenguajes denotados por E_1 y E_2 respectivamente.
2. Luego, puedo **determinizar** ambos autómatas, tal como vimos en la teórica (usando el método de la construcción de subconjuntos), obteniendo M_1' y M_2'.
3. Ahora, puedo **minimizar** ambos autómatas determinísticos, obteniendo M_1'' y M_2''.
   Me estoy valiendo de minimización, que no vimos en la materia, pero sabemos que el lenguaje generado por dos autómatas finitos es el mismo si y solo si los autómatas finitos minimizados son iguales.
4. Finalmente, puedo **comparar los autómatas finitos minimizados** M_1'' y M_2'' para determinar si los lenguajes generados por E_1 y E_2 son iguales.

De manera alternativa propongo,
Si tenés un ER, la podés pasar a AFD; luego, de AFD podés pasarlo a AP determinístico, por lo que es, en particular, un lenguaje libre de contexto determinístico; y en una diapositiva se mencionó que hay un algoritmo hecho por Géraud Sénizergues para saber si dos lenguajes libres de contexto determinísticos son iguales.

También,
Podemos usar la **diferencia simétrica**, pues.

Diferencia simétrica $\rightarrow L_1 \oplus L_2 = (L_1 - L_2) \cup (L_2 - L_1)$.
Donde la **diferencia** $L_1 - L_2$ se puede obtener como $L_1 \cap \overline{L_2}$.

Luego, $E_1 = E_2 \iff \mathcal{L}(E_1) \oplus \mathcal{L}(E_2) = \emptyset$.

10.3. Ejercicio 2

Dar dos algoritmos distintos para determinar si el lenguaje aceptado por un autómata finito dado es el conjunto de todas las cadenas del alfabeto. Justificar cada uno.

Me piden dar dos algoritmos distintos para saber si el lenguaje aceptado por un autómata finito es Σ^*.

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un autómata finito.

Preguntarme si $\mathcal{L}(M) = \Sigma^*$ es equivalente a preguntarme si $\mathcal{L}(M^c) = \emptyset$.
Luego, basta ver que $\mathcal{L}(M^c) = \emptyset$.
Sé que los autómatas finitos son cerrados por complemento.
Si M no era AFD completo, lo hago AFD completo usando el método de construcción de subconjuntos y agrego los estados «fantasmas» que sean necesarios.

10.3.1. Algoritmo 1

Por el **lema de pumping**, sé que $\mathcal{L}(M^c) = \emptyset \iff$ el no existe cadena de longitud menor que n que pertenezca a $\mathcal{L}(M^c)$.
Esto es computable (particularmente r.p.) pues es un existe acotado: palabras de longitud n

con los símbolos del alfabeto finito, a su vez, saber si una palabra pertenece a un lenguaje regular es computable (en tiempo lineal).

10.3.2. Algoritmo 2

Como todo lenguaje aceptado por autómata finito, tiene una gramática regular que lo genera, puedo **construir la gramática regular** G^c que genera $\mathcal{L}(M^c)$.
Luego, puedo obtener el conjunto de no terminales activos de G^c con la función $\text{ObtenerNoTerminalesActivos}(G^c)$ y ver si S peternece al conjunto de terminales activos.

- Si S es un terminal activo $(S \in \text{ObtenerNoTerminalesActivos}(G^c))$, entonces $\mathcal{L}(G^c) \neq \emptyset$.
- Si S no es un terminal activo $(S \notin \text{ObtenerNoTerminalesActivos}(G^c))$, entonces $\mathcal{L}(G^c) = \emptyset$.

10.4. Ejercicio 3

Dar un algoritmo que determine si un lenguaje regular dado es infinito. Justificar.

Sé, por el **lema de pumping**, que L es infinito $\iff$ existe una palabra de longitud entre n y $2n - 1$ que pertenece a L.
Sé que este algoritmo es computable pues es un existe acotado y saber si una palabra pertenece a un lenguaje regular es computable.

10.5. Ejercicio 4

¿Cuántos autómatas finitos deterministas con dos estados pueden construirse sobre el alfabeto $\{0, 1\}$?

Tomo la definición de AFD como $M = \langle Q, \Sigma, \delta, q_0, F \rangle$. Esto quiere decir que,

- Debe haber al menos un estado inicial.
- El conjunto de estados finales puede ser vacío.

Tengo limitada la cantidad de estados a 2 y me dan el alfabeto Σ.
Significa que puedo cambiar 3 cosas de mi autómata finito determinístico:

1. El estado inicial.
2. El conjunto de estados finales.
3. La función de transición.

Para el estado inicial puedo elegir cualquiera de los dos estados de dos maneras.

- Es inicial.
- No es inicial.

Lo que sí, elegir uno me deja fijado el otro.

Para el conjunto de estados finales puedo elegir cualquier subconjunto del conjunto Q, por lo que hay $|\mathcal{P}(Q)| = 2^2 = 4$ posibles conjuntos de estados finales.
Sé que la función de transición va de $Q \times \Sigma \to Q$. $|Q| = 2, |\Sigma| = 2$. Por lo que el número de transiciones posibles es $2^{2\times 2} = 2^4 = 16$.

Luego, el número total de autómatas finitos determinísticos con dos estados es $2 \times 4 \times 16 = 128$.

10.6. Ejercicio 5

Sean L_1 y L_2 lenguajes regulares. Hacer un AFD que reconoce el producto cartesiano de L_1 y L_2.

Es la concatenación pero ligeramente modificada.

Pongo "(" antes de arrancar el autómata que reconoce $L_1 L_2$, luego, en lugar de las transiciones λ para pasar del autómata que reconoce L_1 al que reconoce L_2, pongo una transición con el símbolo "," para finalmente poner una transición que lee ")", de todos los estados finales del autómata que reconoce L_2, a un estado final nuevo q_f, que será el único estado final del autómata.

10.7. Ejercicio 6

Determinar verdadero o falso y justificar.

1. Para cada AF hay infinitos AFD que reconocen el mismo lenguaje
   Verdadero
   Siempre puedo agregar estados inalcanzables.
2. Si L es libre de contexto, todo subconjunto de L es libre de contexto
   Falso
   Contraejemplo: $L = \{a^n b^n c^m \mid n, m \geq 0\}$, un subconjunto es $L' = \{a^n b^n c^n \mid n \geq 0\}$, que no es libre de contexto.
3. El lema de pumping para lenguajes libres de contexto implica que si un lenguaje se puede bombear entonces es regular o libre de contexto.
   Falso
   Libre de contexto implica que se cumple el lema de pumping, no vale la vuelta.
4. Los autómatas finitos determinísticos reconocen una cadena de longitud n en exactamente n transiciones.
   Verdadero
   No hay transiciones λ y es determinístico, en cada transición se lee un símbolo, no me puedo quedar «boludeando» en el autómata, siempre «avanzo».
5. Los autómatas de pila determinísticos reconocen una cadena de longitud n en exactamente n transiciones.
   Falso
   Puedo tener configuraciones ciclantes.
6. Sea M un AFD y sea M^R el autómata que resulta de revertir función de transición. $\mathcal{L}(M) \cap \mathcal{L}(M^R)$ es regular.
   Verdadero
   M^R sigue siendo un autómata finito, por lo que $\mathcal{L}(M^R)$ es regular. Luego, la intersección de dos lenguajes regulares es regular.

10.8. Ejercicio 7

Dado APD $P = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$, dar un algoritmo que decida si $\mathcal{L}(P) = \Sigma^*$. Justificar la correctitud.

Dado que el autómata de pila es determinístico, puedo construir un autómata de pila que acepte el complemento de $\mathcal{L}(P)$, P^c.

Los lenguajes libres de contexto determinísticos están cerrados por complemento, **solamente los determinísticos**.

Asumo construido P^c igual a la demostración vista en clase, que se puede construir, tal como se hace en la demostración.
Luego, puedo determinar si $\mathcal{L}(P^c) = \emptyset$.

Para ello basta con ver:

- si S no es un terminal activo de la gramática regular que genera $\mathcal{L}(P^c)$, que la puedo generar pues todo autómata de pila tiene una gramática que genera el lenguaje que acepta el autómata.
- o, sea n la constante del lema de pumping, si no existe una palabra de longitud menor que n que pertenezca a $\mathcal{L}(P^c)$.

10.9. Ejercicio 8

Determinar verdadero o falso y justificar. Es decidible que:

1. La intersección de dos conjuntos c.e. es un conjunto c.e.
   Verdadero
   Sí, pues el predicado característico de la intersección de dos conjuntos c.e. es computable, agarrás ambos predicados y usás el $\wedge$ r.p.
2. Si HALT fuera computable entonces la complejidad de Kolmogorov $K : \mathbb{N} \to \mathbb{N}$, $K(s) = \text{mín}\{|p| : \Psi^1(p) = s\}$, sería computable.
   Verdadero
   Como los programas son infinitos numerables, y sabemos que hay una "cota trivial superior" ← printf(...) para la complejidad de Kolmogorov, podríamos enumerar todos los programas para luego con HALT ver si terminan, de más chico a más grande.
   Es decir, programa 0 es la instrucción *pass*, ...
   Si no termina lo salteamos, si termina lo simulamos con la máquina de Turing universal y vemos si devuelve s. Si devuelve s, devolvemos $|p|$, si no devuelve s, seguimos con el siguiente programa.
   A su vez, como estamos recorriendo los programas, tal que $|p_i| < |p_j|$ con $i < j$, si en algún momento llegamos a un p_i tal que $|p_i| > |\text{printf}(...)|$, terminamos y devolvemos $|\text{printf}(...)|$.
3. La pertenencia de una palabra a un lenguaje computable es computable.
   Verdadero
   Si el lenguaje es computable, entonces el predicado característico es computable.
4. Clausura de Kleene de un lenguaje c.e. es c.e.
   Verdadero
   Sea L un lenguaje c.e., sé que su predicado característico es parcialmente computable, es decir, tengo una máquina de Turing que acepta cadenas de L. Luego, puedo construir una máquina de Turing que acepte L^n, para todo n. Es como la «concatenación» de máquinas de Turing.
5. Clausura de Kleene de lenguaje computable es computable
   Verdadero
   Idem arriba pero con lenguajes computables.

6. La reversa de un lenguaje computable es computable.
   Verdadero
   La reversa es r.p.
   Tengo el predicado característico de L, computable. Luego L^r tiene como predicado característico $p_L(\text{reversa}(s))$.
7. La reversa de un lenguaje c.e es c.e.
   Verdadero
   Idem arriba, pero reemplazando computable por c.e.
8. Todo conjunto infinito c.e. tiene un subconjunto infinito computable
   Verdadero
   Sabemos que el conjunto c.e. es infinito, por lo que es $\neq \emptyset$.
   Luego, sé que existe una función f r.p. que lo numera, es decir, $L = \{f(0), f(1), f(2), ...\}$.
   Defino un subconjunto D, tal que $D = \{f(i) : f(i) \succ f(j) \text{ para todo } j < i\}$, en otras palabras, $f(i) \in D$ si es más larga que todas las palabras que aparecieron antes que $f(i)$ en la enumeración.

$$\text{Ejemplo, si } L = \{aa, aaa, 0, a, bb, bbb, 11, cccc, 222, ...\},$$
$$D = \{aa, aaa, bbb, cccc, ...\}.$$

   Falta probar que es infinito y computable.
   Digo que es infinito, pues si no lo fuese, significaría que hay una palabra en D con longitud $\geq$ a todas las demás. Sea $f(m)$ el último elemento de D -de mayor longitud-, como $f(m)$ fue el «último record», para todo $i > m$ se tiene $f(i) \succ f(m)$.
   Esto implica que las palabras de L están acotadas por $|f(m)|$, lo que es un absurdo pues L es infinito.
   Para probar que es computable, queremos mostrar que existe un algoritmo que, dada una palabra w, decide (en tiempo finito) si $w \in D$.

Arrancamos con $i = 0$.
Obtenemos $f(i)$.
Vemos si $w = f(i)$.
En caso afirmativo, devolvemos **1**.
En caso negativo, separamos en dos casos:
Si $|w| < |f(i)|$, devolvemos **0**.
Si $|w| > |f(i)|$, incrementamos i y volvemos a empezar.

10.10. Ejercicio 9

Determinar verdadero o falso y justificar:

1. Toda función total de $\mathbb{N}$ en $\mathbb{N}$ es computable.
   Falso
   Hay funciones totales de $\mathbb{N} \to \mathbb{N}$ que no son computables. El conjunto de funciones computables es numerable, mientras que el cardinal del conjunto que tiene todas las funciones totales de $\mathbb{N} \to \mathbb{N}$ es infinito no numerable (argumento diagonal de Cantor).
2. El conjunto de funciones parcialmente computables de $\mathbb{N}$ en $\mathbb{N}$ es c.e.
   Verdadero
   Las funciones parcialmente computables sabemos que son infinitas numerables, pues cada

función parcialmente computable puede ser representada por un programa en S++, condificado con números naturales.
Estos programas en S++ tienen una biyección con los números naturales.
Vimos en clase que la función biyectiva que codifica los programas en S++ como números naturales es computable.
Por lo tanto, el conjunto de funciones parcialmente computables es c.e., ya que existe un procedimiento efectivo *(la función biyectiva)* para listar todos los programas en S++ que las representan.

Estoy utilizando la siguiente definición conjunto computablemente enumerable:

- *«In computability theory, a set S of natural numbers is called computably enumerable (c.e.), recursively enumerable (r.e.), semidecidable, partially decidable, listable, provable or Turing--recognizable if:»*
  - *«There is an algorithm that enumerates the members of S. That means that its output is a list of all the members of S : $s_1, s_2, s_3, ...$ If S is infinite, this algorithm will run forever, but each element of S will be returned after a finite amount of time. Note that these elements do not have to be listed in a particular way, say from smallest to largest.»*

Extraida de Wikipedia: Computably enumerable set°

10.11. Ejercicio 10

Sea APD $P = (Q, \Sigma, \delta, \Gamma, q_0, F)$ y AP $S = (Q', \Sigma', \delta', \Gamma, q_{0'}, F')$.
Determinar verdadero o falso y justificar.

1. $\mathcal{L}(P) \cup \mathcal{L}(S)$ es libre de contexto.
   Verdadero
   La unión de dos lenguajes libres de contexto es libre de contexto.

2. $\mathcal{L}(P) \cap \mathcal{L}(S)$ es libre de contexto.
   Falso
   La intersección de dos lenguajes libres de contexto no necesariamente es libre de contexto.

3. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto que no es recursiva a izquierda.
   Entonces, si $w \in \mathcal{L}(G)$, su árbol de derivación tiene altura menor que $(|w| + 1)(|V_N| + 1)$.
   Verdadero
   Llamemos $n = |w|$ y $k = |V_N|$.
   Consideremos el árbol de derivación más a la izquierda para $S \overset{h}{\underset{L}{\Rightarrow}} w$.
   Sea n_0 un nodo hoja del árbol de derivación con etiqueta $a \in V_T^*$.
   Supongamos que hay un camino de longitud mayor o igual que $(n + 1)(k + 1)$ nodos de la raíz a la hoja n_0. Es decir,
   Sea $S = X_0 \frown X_1 \frown ... \frown X_{(n+1)(k+1)} = a$, el camino desde la raíz a n_0.
   Separemoslos en $n + 1$ segmentos de $k + 2$ nodos,

$$S = X_0 \frown \ldots \frown X_{k+1}$$
$$X_{k+1} \frown \ldots \frown X_{2(k+1)}$$
$$\ldots$$
$$X_{n(k+1)} \frown \ldots \frown X_{(n+1)(k+1)} = a.$$

Consideremos los $n+1$ subárboles de derivación, con $\alpha, \beta \in (V_N \cup V_T)^*$ apropiados.

$$X_0 \underset{L}{\overset{k+1}{\Rightarrow}} \alpha_1 X_{k+1} \beta_1$$
$$X_{k+1} \underset{L}{\overset{k+1}{\Rightarrow}} \alpha_2 X_{2(k+1)} \beta_2$$
$$\ldots$$
$$X_{n(k+1)} \underset{L}{\overset{k+1}{\Rightarrow}} \alpha_{n+1} X_{(n+1)(k+1)} \beta_{n+1}.$$

Es imposible que todas las derivaciones aporten uno o más terminales a w, pues $|w| = n$, y acá hay $n+1$ derivaciones. Luego, al menos una de estas derivaciones aporta λ a w.

Supongamos que fuese la i-ésima derivación. Eso implica que $\alpha_i, \beta_i \in V_N^*$ pues no pueden aportar terminales a w.
En particular, tanto α_i como β_i aportan λ, pues sabemos que

$$X_{i(k+1)} \underset{L}{\Rightarrow} X_{i+1(k+1)} \underset{L}{\overset{*}{\Rightarrow}} \alpha_{n+1} a \beta_{n+1}.$$

Y como en cada derivación hay $k+1$ pasos de derivación pero solamente $|V_N| = k$ no terminales, en algún momento de la i-ésima derivación, se llegó a algo de la pinta $X_i \underset{L}{\overset{+}{\Rightarrow}} X_i$.
Sin embargo, habíamos dicho que G era no recursiva a izquierda, por lo que llegamos a un absurdo.
Luego, la altura del árbol de derivación es menor que $(n+1)(k+1)$, pues no puede haber un camino de longitud mayor o igual que $(n+1)(k+1)$ nodos de la raíz a una hoja.

4. Para todo AP hay un APD que reconoce el mismo lenguaje.
   Falso
   No tienen el mismo poder expresivo.

5. Clausura de Kleene de un lenguaje regular es regular.
   Verdadero
   Me puedo armar el autómata finito. De hecho, creo que es un ejercicio de la práctica.

6. Clausura de Kleene de un lenguaje libre de contexto determinístico es libre de contexto determinístico.
   Falso
   Vimos en la teórica que los lenguajes libres de contexto determinísticos no están cerrados por la clausura de Kleene.

7. Clausura de Kleene de un lenguaje libre de contexto es libre de contexto.
   Verdadero
   Me puedo armar el autómata de pila.

8. Es decidible si dos AFs reconocen el mismo lenguaje.
   Verdadero
   Se puede pasar de AF a ER con el algoritmo que vimos en la práctica, luego, sale de inmediato por el ejercicio 1.

10.12. Ejercicio 11

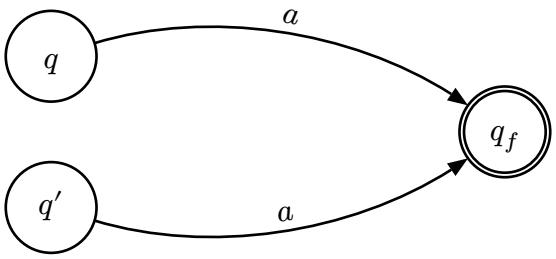
10.12.1. Definición de codeterminismo

Un autómata finito se dice que es **codeterminístico** si, para cualquier par de estados p y q en el autómata, y para cualquier símbolo a en el alfabeto, no existe ningún estado r tal que sea alcanzable desde p con la entrada a como desde q con la misma entrada a (es decir, no existe r tal que $r \in \delta(p, a)$ y $r \in \delta(q, a)$).

Es decir, no puede haber un estado que sea alcanzable por dos estados distintos, leyendo el mismo símbolo del alfabeto.

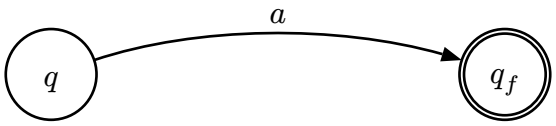
Dar un algoritmo que codeterminice un automata finito.

Intuitivamente lo que queremos es, dado un autómata con transiciones de esta pinta:

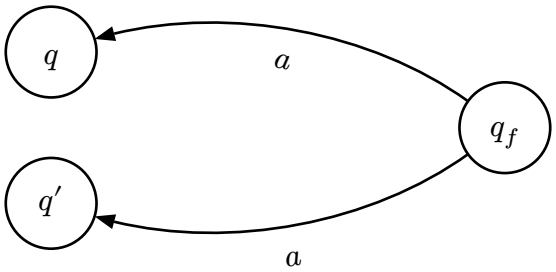


Puedo asumir, sin pérdida de generalidad, que hay un único estado final.

Pasarlo a algo de esta pinta:



Pero fijate esto, tomamos el reverso del autómata del primer dibujo, nos queda algo interesante:



Lo que hizo que nos quede un autómata finito totalmente no determinístico.

Es por ello que propongo la siguiente idea para codeterminizar un autómata finito:
- Hacemos el reverso del autómata finito.
- Determinizamos el autómata finito resultante.
- Tomamos otra vez el reverso del autómata finito.

De esta manera, vamos a quitar las transiciones del estilo $\delta(q_f, a) = \{q, q'\}$ del autómata finito reverso, lo que implica que sacamos las transiciones del estilo $\delta(q, a) = \{q_f\}$ y $\delta(q', a) = \{q_f\}$ del autómata finito original.

Esta idea está basada en el algoritmo de Brzozowski° para minimizar un autómata finito.
La única diferencia entre su algoritmo y este, es que el determiniza otra vez el autómata finito.

10.13. Ejercicio 12

Dado un autómata finito determinístico $A = \langle Q_A, \Sigma, \delta_A, q_0, F_A \rangle$ y dado un autómata de pila determinístico $P = \langle Q_P, \Sigma, \Gamma, \delta_P, p_0, Z, F_P \rangle$, dar un algoritmo que decida si el lenguaje $L(A) \cap L(P)$ es finito. Justificar la correctitud.

Hecho con demostración en la sección 7.7.2.

10.14. Ejercicio 13

Sea Σ un alfabeto. Dada una gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$ que no es recursiva a izquierda y una palabra $w \in V_T^*$, dar un algoritmo que explore los árboles de derivación de G y para determinar si $w \in \mathcal{L}(G)$ o no. Justificar la correctitud.

Se podría hacer BFS (o DFS) en todos los árboles de derivación, tipo fuerza bruta, tal que si la altura del árbol se pasa de $(|w| + 1)(|V_N| + 1)$ y no generó la cadena w aún, entonces rechazamos (backtracking en caso de DFS).
La respuesta del algoritmo sería el $\vee$ de todas las respuestas de los árboles de derivación.
Notar que aunque los árboles de derivación sean infinitos, estos crecen verticalmente, no horizontalmente *(las producciones son finitas)*, por lo que el algoritmo termina *(por nuestra cota de la altura)*.

El algorito es correcto porque es un algoritmo de bruteforce.

10.15. Ejercicio 14

Determinar verdadero o falso y justificar:

1. Si un lenguaje es reconocible por un autómata contador entonces el lenguaje complemento también.
   Falso
   Contraejemplo: $L = \{a^n b^n c \mid n \geq 0\}$ es reconocible por un autómata contador, pero $a^{n+2} b^{n+2} c^{n+2} \notin L$ no es reconocible por un autómata contador (ni siquiera es libre de contexto).
2. Si dos lenguajes L_A y L_B son reconocibles por autómatas contadores entonces el lenguaje de su unión $L_A \cup L_B$ también.
   Verdadero
   Tengo dos autómatas contadores, M_A y M_B, que generan L_A y L_B respectivamente. Puedo armarme un autómata contador M que reconozca $L_A \cup L_B$ de la siguiente manera:
   - Los estados son $Q = Q_A \cup Q_B \cup \{q_0\}$.
   - q_0 estado inicial.
   - Agrego transiciones λ de q_0 a los estados iniciales de M_A y M_B, q_{0_A} y q_{0_B} respectivamente, sin tocar la pila.
   - El alfabeto de la pila es Z_0, I.

- Mantengo los estados finales de M_A y M_B como estados finales de M.
- El alfabeto Σ asumo que es el mismo para M_A y M_B, por lo que es el mismo para M.

Faltaría probar que $\mathcal{L}(M) = L_A \cup L_B$, aunque se puede notar por construcción.

3. Si dos lenguajes L_A y L_B reconocibles por un autómatas contadores, entonces el lenguaje de su intersección $L_A \cap L_B$ también.
   Falso
   Contraejemplo: $L_A = \{a^n b^n c^m \mid n, m \geq 0\}$ y $L_B = \{a^n b^m c^m \mid n, m \geq 0\}$ son reconocibles por autómatas contadores, pero $L_A \cap L_B = \{a^n b^n c^n \mid n \geq 0\}$ no es reconocible por un autómata contador.

Ayuda: Considerar cómo se demuestran y cómo se refutan las propiedades de clausura de los lenguajes libres de contexto.

Notar que el item 1. del complemento se puede justificar tambien como:
Supongamos que $\bar{L}$ es reconocible por un autómata contador.
Entonces, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ sería reconocible por un autómata contador, lo que es un absurdo.

10.16. Ejercicio 15

Dado R un lenguaje regular y dado L un lenguaje libre de contexto determinístico, dar un algoritmo que decide si $L = R$.

Respuesta en el Hopcroft 1976, teorema 10.6.

10.17. Ejercicio 16

Dado R un lenguaje regular y dado L un lenguaje libre de contexto determinístico, ¿es decidible si $R \subseteq L$?

Respuesta en el Hopcroft 1976, teorema 10.6.

10.18. Ejercicio 17

Un autómata de cola es un autómata posiblemente no determinístico similar a un autómata de pila pero tiene una cola en vez de una pila. Formalmente un autómata de cola es $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ donde:
- Q es el conjunto de estados
- Σ es el alfabeto de la cinta entrada
- Γ el el alfabeto de cola
- $\delta : Q \times \Sigma \cup \{\lambda\} \times \Gamma \cup \{\lambda\} \to \mathcal{P}(Q \times \Gamma \cup \lambda)$ es la función de transición
- $q_0 \in Q$ es el estado incial.
- $Z_0 \in \Gamma$ es el símbolo inicial de la cola.
- $F \subseteq Q$ es el conjunto de estados finales.

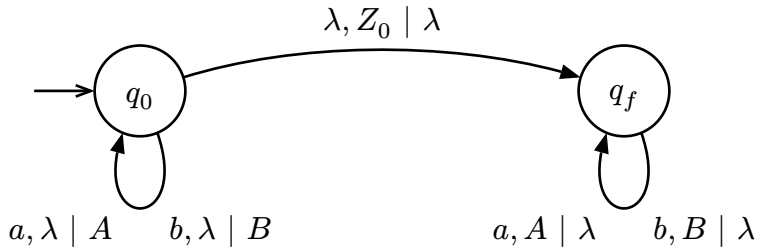
Por ejemplo $\delta(q_1, a, b) = \{(q_2, c), (q_3, d)\}$ dice que el estado q_1 si lee a de la entrada y b está primero en la cola, entonces b sale de la cola, el autómata pasa al estado q_2, y pone c a lo último en la cola.

Demostrar que los autómatas de cola tienen mayor poder expresivo que los autómatas de pila.
Ayuda: dar un lenguaje que no es libre de contexto.

Una forma es demostrar que un autómata de pila puede simular una máquina de Turing, asumiendo que la máquina de Turing tiene mayor poder expresivo que un autómata de pila.

Otra forma menos complicada es la siguiente:
Basta con dar un autómata de cola que reconozca un lenguaje que no sea libre de contexto.
Por ejemplo, el lenguaje $L = \{ww \mid w \in \Sigma^*\}$ no es libre de contexto, se puede demostrar utilizando el lema de pumping.

Sea $\Sigma = \{a, b\}$, el autómata de cola por cola vacía M que reconoce L es el que se muestra en el dibujo.



10.19. Ejercicio 18

Demostrar que:

1. Todos los lenguajes reconocibles por autómatas finitos son reconocibles por autómatas contadores.
   Es definirte un autómata contador a partir del autómata finito, donde no se use la pila.
   ¿Demostración? en la pregunta 2 de la sección 5.1.8.
2. No todos los lenguajes reconocibles por un autómata contador son reconocibles por un autómata finito.
   $L = \{a^n b^n \mid n \geq 0\}$ es reconocible por un autómata contador, pero no es reconocible por un autómata finito. Debería dar el autómata contador que reconoce L y demostrar por el lema de pumping que no es reconocible por un lenguaje regular.