Teoría de Lenguajes

Gianfranco Zambonni

17 de febrero de 2023

Índice general

Capítulo 1

La que da Julio

1.1. Introducción

1.1.1. Relaciones

Dados dos conjuntos A y B, se llama **relación** $R : A \to B$ de A en B a todo subconjutno de $A \times B$, es decir $R \subset A \times B$.

Dos elementos $a \in A$ y $b \in B$ están relacionados si $(a, b) \in R$ y lo notamos aRb.

Si $A = B$, se dice que R es una relación sobre A y se dice que:

- es **reflexica** cuando $\forall a,\ aRa$.
- es **simétrica** cuando $\forall a, b \in A,\ aRb \implies bRa$.
- es **transitiva** cuando $a, b, c \in A,\ aRb \ \wedge\ bRc \implies aRc$.

Relación de equivalencia: Una relación $R : A \to A$ es de **equivalencia** cuando es reflexiva, simétrica y transitiva. Este tipo de relaciones particiona a A en subconjuntos disjuntos llamados **clases de equivalencia**.

Operaciones

Composición de relaciones: Si $R : A \to B$ y $S : B \to C$ son relaciones, entonces la composición de R y S es la relación $S \circ R : A \to C$ definida por:

$$S \circ R = \{(a, c) \mid a \in A,\ c \in C : \exists b \in B,\ aRb \ \wedge\ bSc\}$$

Relación de identidad: La relación de identidad sobre A es la relación $id_A : A \to A$ definida por: $id_A = \{(a, a) \mid a \in A\}$.

- La relación de identidad el el elemento neutro de la composición de relaciones.

Relación de potencia: Dado $R : A \to A$ se define la relación de potencia $R^k : A \to A$ como la composición de k copias de R:

$$R^n = \begin{cases} id_A & \text{si } n = 0 \\ R \circ R^{n-1} & \text{si } n > 0 \end{cases}$$

Clausura transitiva/positiva: Dada una relación $R : A \to A$ se define la clausura transitiva de R como la relación R^+ definida por:

$$R^+ = \bigcup_{n=1}^{\infty} R^n$$

La clausura transitiva de R cumple las siguientes propiedades:

1. $R \subseteq R^+$

2. R^+ es transitiva

DEMOSTRACIÓN

Si aR^+b entonces existe una secuencia de elementos $a = a_0, a_1, \ldots, a_n = b$ tales que $a_i R a_{i+1}$ para todo $i \in [0, n-1]$.

Análogamente, como bR^+c existe una secuencia de elementos

$$b = b_0, b_1, \ldots, b_m = c$$

tales que $b_i R b_{i+1}$ para todo $i \in [0, m-1]$.

Entonces $aR^{n+m}c$ pues puedo armar la secuencia

$$a = a_0, a_1, \ldots, a_n, b_1 \ldots b_m = c$$

.

Luego como $R^{n+m} \subseteq R^+$ vale que aR^+c.

3. Para toda relación $G : A \to A$ tal que $R \subseteq G \wedge G$ es transitiva, entonces $R^+ \subseteq G$, es decir R^+ es la relación transitiva más pequeña que contiene a R.

DEMOSTRACIÓN

Si aR^+b entonces existe una secuencia de elementos $a = a_0, a_1, \ldots, a_n = b$ tales que $a_i R a_{i+1}$ para todo $i \in [0, n-1]$.

Como $R \subseteq G$ entonces $a_i G a_{i+1}$ para todo $i \in [0, n-1]$. Como G es transitiva entonces la aplicación repetida de la transitividad nos lleva a que $a_1 G a_n$, por lo que aGb.

Clausura transitiva reflexiva:

$$R^* = R^+ \cup id_A = \bigcup_{n=0}^{\infty} R^n$$

Observaciones:

- Si A es un conjunto finito, entonces todas las relaciones $R : A \to A$ son finitas.
- Si R es reflexiva, entonces $R^* = R^+$.

1.1.2. Alfabetos

Alfabeto: Un alfabeto es un conjunto finito de símbolos.

Cadena: Una cadena sobre un alfabeto Σ es una secuencia finita de símbolos de Σ. Los símbolos son notados respetando el orden de la secuencia.

Concatenación: Es una operación entre un símbolo del alfabeto Σ y una cadena sobre dicho alfabeto:

$$\circ : \Sigma \times \{\text{cadenas sobre } \Sigma\} \to \{\text{cadenas de } \Sigma\}$$

- La cadena nula λ es el elemento neutro de la concatenación.

Clausura de Kleene de Σ: Σ^*

- $\lambda \in \Sigma^*$
- $\alpha \in \Sigma^* \implies \forall\ a \in \Sigma,\ a \circ \alpha \in \Sigma^*$

Clausura positiva de Σ: $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$

1.1.3. Lenguajes

Lenguaje: Un lenguaje es un conjunto de cadenas sobre un alfabeto Σ.

Concatenación de lenguajes: Si L_1 y L_2 son lenguajes definidos sobre los alfabetos Σ_1 y Σ_2 respectivamente, entonces la concatenación de L_1 y L_2 es un lenguaje L_1L_2 sobre el alfabeto $\Sigma_1 \cup \Sigma_2$ dfinido de la siguiente manera:

$$L_1L_2 = \{\alpha\beta \mid \alpha \in L_1,\ \beta \in L_2\}$$

Clausura de Kleene L^*:

$L^0 = \{\lambda\}$

$L^n = LL^{n-1}$ para $n >= 1$

$L^* = \bigcup_{n=0}^{\infty} L^n$

Clausura positiva L^+:

$L^+ = \bigcup_{n=1}^{\infty} L^n$

Observaciones:

- $L^+ = LL^*$
- $L^* = L^+ \cup \{\lambda\}$
- Si L es un lenguaje definido sobre Σ entonces $L \subseteq \Sigma^*$

1.1.4. Gramáticas

Una gramática es una 4-tupla (V_N, V_T, P, S) donde:

- V_N es un conjunto finito de símbolos no terminales.
- V_T es un conjunto finito de símbolos terminales.
- P es un conjunto finito de reglas de producción: Son pares ordenados $\alpha \to \beta$ donde

$$\alpha \in (V_N \cup V_T)^* V_N (V_N \cup V_T)^* \text{ y } \beta \in (V_N \cup V_T)^*$$

- $S \in V_N$ es el símbolo inicial.

Dada una producción $A \to \alpha \in P$, se denomina a A como **cabeza** de la producción y a α como su **cuerpo**.

Derivación: El el proceso por el cual se obtiene una cadena a partir de un símbolo inicial remplazando recursivamente símbolos no terminales por cuerpos de producciones en P cuya cabeza coincida con los símbolos que están siendo remplazados.

Forma setencial de una grámatica: Se llama forma sentencial a cualquier derivación de la grámatica:

- S es una forma setencial de G
- Si $\alpha\beta\gamma$ es una forma setencial de G y $\beta \to \delta \in P$ entonces $\alpha\delta\gamma$ es una forma setencial de G.

Derivación directa en G: Si $\alpha\beta\gamma \in (V_N \cup V_T)^*$ y $\beta \to \delta \in P$ entonces $\alpha\delta\gamma$ es una derivación directa de G de $\alpha\beta\gamma$ y se denota como $\alpha\beta\gamma \underset{G}{\Longrightarrow} \alpha\delta\gamma$.

- $\overset{+}{\underset{G}{\Longrightarrow}}$ es la clausura positiva.
- $\overset{*}{\underset{G}{\Longrightarrow}}$ es la clausura transitiva y reflexiva.
- $\overset{k}{\underset{G}{\Longrightarrow}}$ será la potencia k-ésima.

Lenguaje de una grámatica $\mathcal{L}(G)$: Es el conjunto de todas las cadenas de símbolos terminales que son formas setenciales de G.

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

Clasificación de grámaticas (Chomsky)

Gramáticas regulares (tipo 3): Son aquellas gramáticas que cumplen alguna de las siguientes condiciones:

- Todas sus producciones son de la forma $A \to aB$ ó $A \to a$ ó $A \to \lambda$ donde $A, B \in V_N$ y $a \in V_T$. En este caso se dice que es una gramática lineal a derecha.
- Todas sus producciones son de la forma $A \to Ba$ ó $A \to a$ ó $A \to \lambda$ donde $A, B \in V_N$ y $a \in V_T$. En este caso se dice que es una gramática lineal a izquierda.

Gramáticas libres de contexto (tipo 2): Son aquellas gramáticas en las que cada producción es de la forma $A \to \alpha$ donde $A \in V_N$ y $\alpha \in (V_N \cup V_T)^*$.

De la definición anterior puede inferirse que toda grámatica regular es libre de contexto.

Gramáticas sensibles al contexto (tipo 1): Son aquellas gramáticas en las que cada producción es de la forma $\alpha \to \beta$ donde $\alpha, \beta \in (V_N \cup V_T)^*$ y $|\alpha| \leq |\beta|$. Se puede inferir que toda gramática independiente del contexto que no posea regla borradoraas (es decir, que no posea producciones de la forma $A \to \lambda$) es sensible al contexto.

Gramáticas sin restricciones (tipo 0): Son aquellas gramáticas que no poseen ninguna restricción sobre la forma de sus producciones. El conjunto de las grámaticas tipo 0 es el conjunto de todas las grámaticas y permite generar todos los lenguajes aceptados por una máquina de Turing. También se los conoce como lenguajes recursivamente enumerables (r.e)

Definición: Un lenguaje generado por una grámatica tipo t es llamado **lenguaje tipo** t.

1.2. Autómatas finitos

1.2.1. Autómatas finitos deterministicos (AFD)

Un autómata finito determinista es una 5-tupla $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ donde:

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

Función de transición generalizada $\hat{\delta}$: La función de transición δ está definida para que tome como parámetro un único símbolo de Σ. Se puede extender para que tome como parámetro una cadena de símbolos de Σ, es decir $\hat{\delta} : Q \times \Sigma^* \to Q$:

- $\hat{\delta}(q, \lambda) = q$
- $\hat{\delta}(q, \beta a) = \delta(\hat{\delta}(q, \beta), a)$ con $\beta \in \Sigma^*$ y $a \in \Sigma$

Cadena aceptada por un AFD: Una cadena $\beta \in \Sigma^*$ es aceptada por un AFD $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ si y solo si $\hat{\delta}(q_0, \beta) \in F$.

Lenguaje aceptado por un AFD: El lenguaje aceptado por un AFD $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ es el conjunto de todas las cadenas $\beta \in \Sigma^*$ que son aceptadas por $\mathcal{M}$:

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

1.2.2. Autómatas finitos no deterministas (AFND)

Un autómata finito no determinista es una 5-tupla $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ donde:

- Q es un conjunto finito de estados.
- Σ es un conjunto finito de símbolos de entrada.
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ es una función de transición.

  A diferencia de los AFD, la función δ devuelve un conjunto de estados en lugar de un solo estado.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

Función de transición generalizada $\hat{\delta}$: Primero vamos a definir $\delta_P : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$ de la siguiente manera:

$$\delta_P(P, a) = \bigcup_{p \in P} \delta(p, a)$$

La función $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$ se define de manera recursiva como:

- $\hat{\delta}(q, \lambda) = \{q\}$
- $\hat{\delta}(q, \beta a) = \{p : \ \exists r \in \hat{\delta}(q, \beta) \text{ tal que } p \in \delta(r, a)\} = \delta_P(\hat{\delta}(q, \beta), a)$ con $\beta \in \Sigma^*$ y $a \in \Sigma$

Para generalizar a un más podemos definir $\hat{\delta}_P : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ de la siguiente manera:

$$\hat{\delta}_P(P, \beta) = \bigcup_{q \in P} \hat{\delta}(q, \beta)$$

Cadena aceptada por un AFND: Una cadena $\beta \in \Sigma^*$ es aceptada por un AFND $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ si y solo si $\hat{\delta}(q_0, \beta) \cap F \neq \emptyset$. Es decir, si alguno de los estados alcanzados por $\hat{\delta}(q_0, \beta)$ es un estado final.

Lenguaje aceptado por un AFND: El lenguaje aceptado por un AFND $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ es el conjunto de todas las cadenas $\beta \in \Sigma^*$ que son aceptadas por $\mathcal{M}$:

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

Equivalencia entre AFD y AFND

Es trivial ver que para todo AFD existe un AFND que acepte el mismo lenguaje.

Teorema 1.2.1. Dado una AFND $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$, existe un AFD $\mathcal{M}' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ tal que $L(\mathcal{M}) = L(\mathcal{M}')$.

Vamos a demostrar este teorema construyendo una AFD $\mathcal{M}'$ a partir de $\mathcal{M}$. Una vez constuido deberemos demostrar que $\mathcal{M}'$ acepta el mismo lenguaje que $\mathcal{M}$.

Construcción de $\mathcal{M}'$:

- Q' será el conjunto de partes $\mathcal{P}(Q)$. Vamos a denotar cada estado $s \in Q'$ con etiquetas del estilo $[q_1, \ldots, q_k]$ donde $q_1, \ldots, q_k \in Q$. Entonces:

$$Q' = \mathcal{P}(Q)$$

- $\delta'([q_1, \ldots, q_k], a) = [p_1, \ldots, p_m] \iff \delta_P(\{q_1, \ldots, q_k\}, a) = \{p_1, \ldots, p_m\}$
- $q_0' = [q_0]$
- $F' = \{[q_1, \ldots, q_n] \in Q' : \{q_1, \ldots, q_n\} \cap F \neq \emptyset\}$

Equivalencia entre funciones de transición: Antes de demostrar que ambos automátas aceptan el mismo lenguaje, vamos a demostrar que las funciones de transición generalizadas de ambos automátas son equivalentes cuando las llamamos con el estado inicial como primer parámetro. Es decir, queremos ver que $\hat{\delta}'(q_0', \beta) = [p_1, \ldots, p_k] \iff \hat{\delta}(q_0, \beta) = \{p_1, \ldots, p_k\}$.

Lo vamos a hacer por inducción. Recordemos que $q_0' = [q_0]$:

- Caso base: $\beta = \lambda$:
  - $\hat{\delta}'([q_0], \lambda) = [q_0]$ por definición de $\hat{\delta}'$.
  - $\hat{\delta}(q_0, \lambda) = \{q_0\}$ por definición de $\hat{\delta}$.

  Luego $\hat{\delta}'([q_0], \lambda) = [q_0] \iff \hat{\delta}(q_0, \lambda) = \{q_0\}$.
- Caso inductivo: $\beta \implies \beta a$:

  Nuestra hipotesis inductiva es $\hat{\delta}'(q_0', \beta) = [r_1, \ldots, r_m] \iff \hat{\delta}(q_0, \beta) = \{r_1, \ldots, r_m\}$.

  Queremos ver que $\hat{\delta}'(q_0', \beta a) = [p_1, \ldots, p_k] \iff \hat{\delta}(q_0, \beta a) = \{p_1, \ldots, p_k\}$

$$
\begin{aligned}
\hat{\delta}'(q_0', \beta a) = [p_1, \ldots, p_k] &\underset{\text{def.}}{\iff} \delta'(\hat{\delta}'(q_0', \beta), a) = [p_1, \ldots, p_k] \\
&\underset{\text{def.}}{\iff} \exists [r_1, \ldots, r_m] \in Q' \text{ tal que } \delta'(q_0', \beta) = [r_1, \ldots, r_m] \\
&\qquad \wedge \delta'([r_1, \ldots, r_m], a) = [p_1, \ldots, p_k] \\
&\underset{\substack{\text{H.I} \\ \text{constr.}}}{\iff} \exists \{r_1, \ldots, r_m\} \in Q \text{ tal que } \hat{\delta}(q_0, \beta) = \{r_1, \ldots, r_m\} \\
&\qquad \wedge \delta_P(\{r_1, \ldots, r_m\}, a) = \{p_1, \ldots, p_k\} \\
&\underset{\text{def.}}{\iff} \delta_P(\hat{\delta}(q_0, \beta), a) = \{p_1, \ldots, p_k\} \underset{\text{def.}}{\iff} \hat{\delta}(q_0, \beta a) = \{p_1, \ldots, p_k\}
\end{aligned}
$$

Demostración de la equivalencia: Ahora que hemos demostrado que las funciones de transición generalizadas de ambos automátas son equivalentes, vamos a demostrar que ambos automátas aceptan el mismo lenguaje:

$$
\begin{aligned}
\beta \in \mathcal{L}(\mathcal{M}) &\underset{\text{def.}}{\iff} \hat{\delta}(q_0, \beta) = \{q_1, \ldots, q_n\} \wedge \{q_1, \ldots, q_n\} \cap F \neq \emptyset \\
&\underset{\substack{\text{equiv.} \\ \text{constr.}}}{\iff} \hat{\delta}(q_0', \beta) = [q_1, \ldots, q_n] \wedge [q_1, \ldots, q_n] \in F' \\
&\underset{\text{def.}}{\iff} x \in \mathcal{L}(M')
\end{aligned}
$$

1.2.3. Autómatas finitos no deterministico con transiciones λ

Un autómata finito no determinista con transiciones λ es un autómata finito no determinista que tiene transiciones λ. Estas transacciones nos permiten ir de un estado a otro sin consumir ningún símbolo de entrada.

Los definimos con una 5-upla $(Q, \Sigma, \delta, q_0, F)$ donde:

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

Clausura λ de un estado q: Se denota $Cl_\lambda(q)$ es el conjunto de estados que se pueden alcanzar desde q siguiendo solo transiciones λ. Es decir,

$$Cl_\lambda(q) = \delta(q, \lambda)$$

Además $q \in Cl_\lambda(q)$.

Clausura λ de un conjunto de estados P:

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

Generalización de la función de transición: Podemos extender δ a conjunto de estados:

$$\delta_P : \mathcal{P}(Q) \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q)$$
$$\delta_P(P, a) = \bigcup_{p \in P} \delta(p, a)$$

Entonces podemos definir:

$$\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$$
$$\hat{\delta}(q_0, \lambda) = Cl_\lambda(q_0)$$
$$\hat{\delta}(q_0, \beta a) = Cl_{P\lambda}\left(\delta_P(\hat{\delta}(q_0, \beta), a)\right) = Cl_{P\lambda}\left(\left\{p : \exists q \in \hat{\delta}(q_0, \beta) \text{ tal que } p \in \delta(q, a)\right\}\right)$$

Tambien extendemos $\hat{\delta}$ a conjuntos de estados:

$$\hat{\delta}_P : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$$
$$\hat{\delta}_P(P, \beta a) = \bigcup_{p \in P} \hat{\delta}(p, \beta a)$$

Cadena aceptada por un AFND-λ: Una cadena β es aceptada por un AFND-λ $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ si y solo si $\hat{\delta}(q_0, \beta) \cap F \neq \emptyset$.

Lenguaje aceptado por un AFND-λ: El lenguaje aceptado por un AFND-λ $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ es el conjunto de todas las cadenas aceptadas por M:

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

Equivalencia entre AFND y AFND-λ

Dado un AFND-λ $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ podemos construir un AFND $M' = \langle Q, \Sigma, \delta', q_0, F'\rangle$ tal que M acepte el mismo lenguaje que M'.

Construcción de M': Notemos que ambos autómatas tiene el mismo conjunto de estados Q, el mismo conjunto de símbolos de entrada Σ y el mismo estado inicial q_0. Por lo que solo debemos definir δ' y F'.

- $\delta'(q,a) = \hat{\delta}(q,a) = Cl_{P\lambda}\left(\delta_P(\hat{\delta}(q,\lambda),a)\right)$
- $F' = \begin{cases} F \cup \{q_0\} & \text{si } Cl_\lambda(q_0) \cap F \neq \emptyset \\ F & \text{si no} \end{cases}$

Equivalencia de funciones de transición generalizada: Vamos a demostrar por inducción que $\hat{\delta}'(q_0,\beta) = \hat{\delta}(q_0,\beta)$ para todo $|\beta| \geq 1$:

- Caso base: $|\beta| = 1$. Sea $\beta = a$, entonces $\hat{\delta}'(q_0,\beta) = \hat{\delta}'(q_0,a) = \hat{\delta}(q_0,a)$ por como definimos δ'.
- Caso inductivo: Supongamos que $\hat{\delta}'(q_0,\beta) = \hat{\delta}(q_0,\beta)$ para todo $|\beta| \leq n$. Sea $\omega = \beta a$. Entonces:

$$\hat{\delta}'(q_0,\omega) = \hat{\delta}'(q_0,\beta a) \underset{\text{def.}}{=} \delta'_P(\hat{\delta}'(q_0,\beta),a) \underset{\text{H.I}}{=} \delta'_P(\hat{\delta}(q_0,\beta),a) \qquad (1)$$

Por otro lado, dado $P \subseteq Q$ tenemos que:

$$\delta'_P(P,a) \underset{\text{def.}}{=} \bigcup_{p\in P} \delta'(p,a) \underset{\text{constr.}}{=} \bigcup_{p\in P} \hat{\delta}(p,a) \underset{\text{def}}{=} \hat{\delta}_P(P,a)$$

Entonces, remplazando el último término en (1), nos queda:

$$\delta'_P(\hat{\delta}(q_0,\beta),a) = \hat{\delta}_P(\hat{\delta}(q_0,\beta),a) \underset{\text{def.}}{=} \hat{\delta}(q_0,\beta a) \underset{\text{def.}}{=} \hat{\delta}(q_0,\omega)$$

Demostración de equivalencia: Veamos ahora que M acepta el mismo lenguaje que M', vamos a separar la demostración en dos casos: $\beta = \lambda$ y $\beta \neq \lambda$.

- $\beta = \lambda$
  - $\lambda \in \mathcal{L}(M) \implies \lambda \in \mathcal{L}(M')$

$$\begin{aligned} \lambda \in \mathcal{L}(M) &\underset{\text{def.}}{\iff} \hat{\delta}(q_0,\lambda) \cap F \neq \emptyset \\ &\underset{\text{def.}}{\iff} Cl_\lambda(q_0) \cap F \neq \emptyset \\ &\underset{\text{constr.}}{\implies} q_0 \in F' \underset{\text{def.}}{\iff} \lambda \in \mathcal{L}(M') \end{aligned}$$

- $\lambda \in \mathcal{L}(M') \implies \lambda \in \mathcal{L}(M)$.

$$\begin{aligned}
\lambda \in \mathcal{L}(M') &\underset{\text{def.}}{\implies} q_0 \in F' \\
&\underset{\text{constr.}}{\implies} q_0 \in F \vee Cl_\lambda(q_0) \cap F \neq \emptyset \\
&\underset{\text{def. F}}{\implies} Cl_\lambda(q_0) \cap F \neq \emptyset \vee Cl_\lambda(q_0) \cap F \neq \emptyset \\
&\underset{\text{def.}}{\implies} Cl_\lambda(q_0) \cap F \neq \emptyset \\
&\underset{\text{def.}}{\iff} \lambda \in \mathcal{L}(M)
\end{aligned}$$

- $\beta \neq \lambda$
  - $\beta \in \mathcal{L}(M) \implies \beta \in \mathcal{L}(M')$

$$\begin{aligned}
\beta \in \mathcal{L}(M) &\underset{\text{def.}}{\implies} \hat{\delta}(q_0, \beta) \cap F \neq \emptyset \\
&\underset{\text{equiv. tran.}}{\implies} \hat{\delta}'(q_0, \beta) \cap F \neq \emptyset \\
&\underset{\text{constr. M'}}{\implies} \hat{\delta}'(q_0, \beta) \cap F' \neq \emptyset \underset{\text{def.}}{\implies} \beta \in \mathcal{L}(M')
\end{aligned}$$

  - $\beta \in \mathcal{L}(M') \implies \beta \in \mathcal{L}(M)$

$$\begin{aligned}
\beta \in \mathcal{L}(M') &\underset{\text{def.}}{\implies} \hat{\delta}'(q_0, \beta) \cap F' \neq \emptyset \\
&\underset{\text{equiv. trans}}{\implies} \hat{\delta}(q_0, \beta) \cap F \neq \emptyset \vee \hat{\delta}(q_0, \beta) \cap (F \cup \{q_0\}) \neq \emptyset \hat{\delta}(q_0, \beta) \cap \\
&\underset{\text{constr.} M'}{\implies} \hat{\delta}(q_0, \beta) \cap F \neq \emptyset
\end{aligned}$$

    Si vale la primera parte de la última expresión $\delta(q_0, \beta) \cap F \neq \emptyset$ entonces $\beta \in \mathcal{L}(M)$ por definición.

    Veamos que pasa si vale $\hat{\delta}(q_0, \beta) \cap (F \cup \{q_0\}) \neq \emptyset$, es decir hay un camino de transiciones λ desde q_0 hasta algún estado estado $q' \in F$:

$$\hat{\delta}(q_0, \beta) \cap (F \cup \{q_0\}) \neq \emptyset \implies \hat{\delta}(q_0, \beta) \cap F \neq \emptyset \vee \hat{\delta}(q_0, \beta) \cap \{q_0\} \neq \emptyset$$

    La primer parte es lo mismo que arriba, analizemos la segunda:

$$\hat{\delta}(q_0, \beta) \cap \{q_0\} \neq \emptyset \implies Cl_\lambda(q_0) \cap F \neq \emptyset \implies \lambda \in \mathcal{L}(M)$$

Queda demostrada la equivalencia de lenguajes.

1.3. Expresiones regulares

Una expresión regular es una expresión que describe un lenguaje de forma compacta y sencilla:

- $\varnothing$ es una expresión regular que describe el lenguaje vacío $\emptyset$.
- λ es una expresión regular que describe el lenguaje unitario $\{\lambda\}$.
- Para cada $a \in \sigma$, a es una expresión regular que describe el lenguaje $\{a\}$.
- Si r y s son dos expresiones que denotan los lenguajes R y S entonces:
  - $r|s$ ó $r + s$ es una expresión regular que describe el lenguaje $R \cup S$.
  - rs es una expresión regular que describe el lenguaje RS.
  - r^* es una expresión regular que describe el lenguaje R^*.
  - r^+ es una expresión regular que describe el lenguaje R^+.

Expresiones regulares recursivas: Si $r = \alpha r + \beta$, entonces $r = \alpha^*\beta$. Además, si $\alpha^* = \alpha^+$, entonces $r = \alpha^*(\beta + \gamma)$ para culquier expresión regular γ.

1.3.1. Expresiones regulares a AFND-λ

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

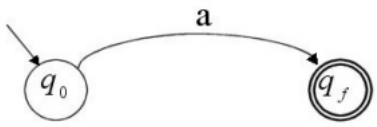
Vamosa a demostrar por inducción sobre los operadores de las expresiones regulares.

Casos base

$r = \varnothing$

$r = \lambda$

$r = a$



Pasos inductivos

Sean r_1 y r_2 dos expresiones regulares. Supongamos que existen AFND-λ $M_1 = \langle Q_1, \Sigma_1, \delta_1, q_1, \{f_1\}\rangle$ y $M_2 = \langle Q_2, \Sigma_2, \delta_2, q_2, \{f_\}\rangle$ tal que $\mathcal{L}(M_1) = \mathcal{L}(r_1)$ y $\mathcal{L}(M_2) = \mathcal{L}(r_2)$. Vamos a armar a partir de estos autómata uno nuevo que acepte los lenguajes generados por las expresiones $r_1|r_2$, r_1r_2, r_1^* y r^+.

$r_1|r_2$: Podemos construir un automata $M_0 = \langle Q_0, \Sigma_0, \delta_0, q_0, \{f_0\}\rangle$ tal que $\mathcal{L}(M_0) = \mathcal{L}(r_1|r_2)$ de la siguiente forma:

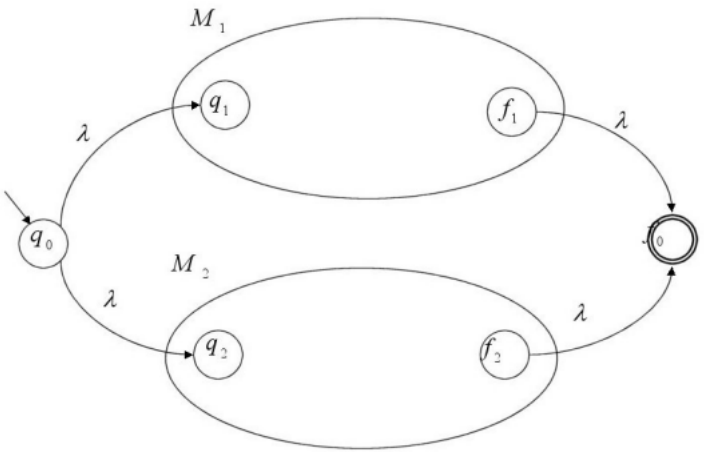
- $Q_0 = Q_1 \cup Q_2 \cup \{q_0, f_0\}$
- $\Sigma_0 = \Sigma_1 \cup \Sigma_2$
- $\delta_0 : Q_0 \times \Sigma_0 \to \mathcal{P}(Q_0)$

  $\delta(q_0, \lambda) = \{q_1, q_2\}$

  $\delta(q, a) = \delta_1(q, a)$ 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 $a \in \Sigma_2 \cup \{\lambda\}$

  $d(f_1, \lambda) = d(f_2, \lambda) = \{f_0\}$



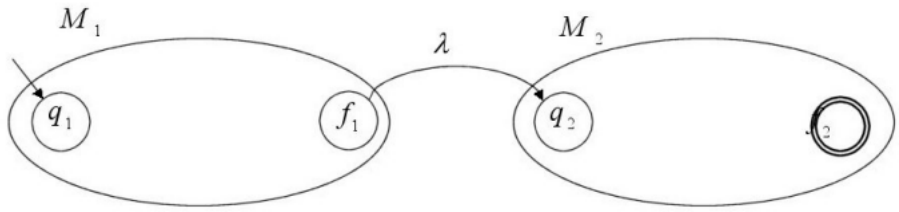
$r_1 r_2$: $M_0 = \langle Q_0, \Sigma_0, \delta_0, q_1, \{f_2\}\rangle$:

- $Q_0 = Q_1 \cup Q_2$
- $\Sigma_0 = \Sigma_1 \cup \Sigma_2$
- $\delta_0 : Q_0 \times \Sigma_0 \to \mathcal{P}(Q_0)$

  $\delta(q, a) = \delta_1(q, a)$ 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 $a \in \Sigma_2 \cup \{\lambda\}$

  $d(f_1, \lambda) = \{q_2\}$

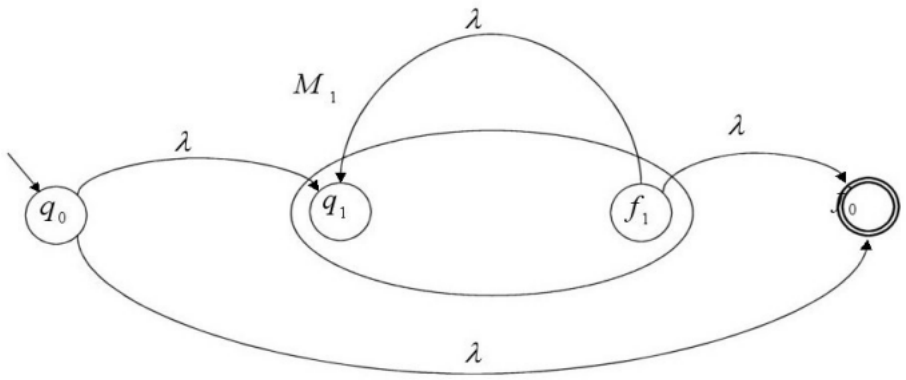


r_1^*: $M_0 = \langle Q_0, \Sigma_1, \delta_0, q_0, \{f_0\}\rangle$:

- $Q_0 = Q_1 \cup \{f_0, q_0\}$
- $\delta_1 : Q_0 \times \Sigma_1 \to \mathcal{P}(Q_0)$

  $\delta(q_0, \lambda) = \delta(f_1, \lambda) = \{q_1, f_0\}$

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



Para el caso r_1^+ es el mismo autómata que para este caso sin la transición $q_0 \xrightarrow{\lambda} f_0$.

1.3.2. AFD a expresión regular

Dado un AFD $M = \langle \{q_1, \ldots, q_n\}, \Sigma, \delta, q_1, F \rangle$, que acepta el lenguaje $\mathcal{L}$, existe una expresión regular que denota el mismo lenguaje.

Demostración

Nombremos $R^k_{i,j}$ a la expresión regular cuyo lenguaje $\omega \subseteq \Sigma^*$ son las cadenas que llevan al autómata M desde el estado q_i al estado q_j pasando solo por estados q_l con $l \leq k$. En particular $R^n_{i,j}$ es la expresión regular que representa todas las cadenas que permiten ir del estado i al estado j.

Vamos a buscar como construir $R^k_{i,j}$ para cada $k \in \{0, \ldots, n\}$ de manera inductiva. Suponiendo que demostramos la existencia de esta expresión regular, podemos concluir que la unión $R^n_{1,f_1}|R^n_{1,f_2}|...|R^n_{1,f_m}$ (con $f_1 \ldots f_m \in F$) es la expresión regular que representa el lenguaje $\mathcal{L}$:

Caso base ($k = 0$): Como todos los estados están enumerados del 1 para arriba, $k = 0$ significa que no debe haber estados intermedios en el camino entre q_i y q_j, por lo que pueden ser de dos formas:

- Una arco del estado i al estado j.
- Un camino de longitud cero que solo contiene el estado i.

Si $i \neq j$, entonces solo es posible la primera opción. Debemos examinar el AFD y encontrar aquellos simbolos que nos permitan ir del estado i al estado j.

1. Si no existe tal símbolo, entonces $R^0_{i,j} = \varnothing$.
2. Si existe exactamente un símbolo a, entonces $R^0_{i,j} = a$.
3. Si existen más de un símbolo, entonces $R^0_{i,j} = a_1|a_2|...|a_n$.

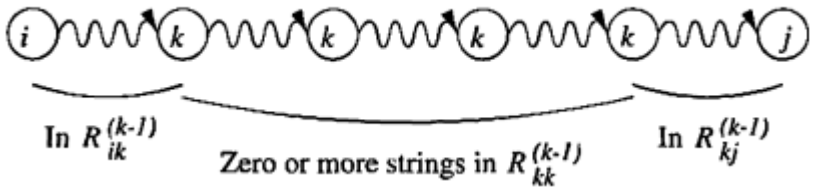
Ahora, si $i = j$ entonces los caminos de longitud cero también son posibles, por lo que habría que agregar a cada una de las expresiones recién mencionadas el simbolo λ:

1. $R^0_{i,j} = \lambda$.
2. $R^0_{i,j} = a|\lambda$.
3. $R^0_{i,j} = a_1|a_2|...|a_n|\lambda$.

Paso inductivo: Supongamos que hay un camino desde el estado i al estado j que no pasa por estados mas grandes k. Entonces podemos considerar las siguientes dos opciones:

1. El camino no pasa por el estado k, por lo que el lenguaje de $R^{k-1}_{i,j}$ contiene a ese camino.

2. El camino pasa por el estado k por lo menos una vez. Entonces podemos partir el camino en varias partes:



La primer parte, va desde el estado i al estado k sin pasar por k, la última parte es desde el estado k al estado j sin pasar por k, y todas las partes intermedia s van desde el estado k al estado k sin pasar por k. Cada una de estas partes ya tiene una expresión regular asociada: $R_{i,k}^{k-1}$, $R_{k,k}^{k-1}$, $R_{k,j}^{k-1}$, por lo que podemos unirlas para obtener la expresión regular que representa el camino completo de la siguiente forma:

$$R_{i,k}^{k-1}\left(R_{k,k}^{k-1}\right)^{*}R_{k,j}^{k-1}$$

Entonces $R_{i,j}^{k}$ es la unión de las expresiones de los dos tipos de caminos que acabamos de describir:

$$R_{i,j}^{k}=R_{i,j}^{k-1}\cup R_{i,k}^{k-1}\left(R_{k,k}^{k-1}\right)^{*}R_{k,j}^{k-1}$$

Finalmente, si construimos en orden todas estas expresiones regulares desde $R_{i,j}^{0}$, eventualmente llegaremos hasta $R_{i,j}^{n}$.

Y como dijimos, más arriba si calculamos $R_{1,j}^{0}$ para cada $q_j \in F$ y unimos todas las expresiones, obtendremos la expresión regular que representa el lenguaje $\mathcal{L}$.

1.3.3. Gramática regular a AFND

Dada una grámatica regular $G=\langle V_N, V_T, P, S\rangle$, podemos construir un AFND $M=\langle Q, \Sigma, \delta, q_0, F\rangle$ que reconozca el lenguaje generado por G

Demostración

Vamos a constuir el autómata finito no determinista M y demostrar que reconoce el lenguaje generado por G.

Construcción de M: Construyamos M de la siguiente manera:

- $Q = V_N \cup \{q_f\}$. Denotarmeos q_A al estado que representa al no símbolo no terminar A.
- $\Sigma = V_T$
- $q_0 = q_S$

- Si $A, B \in V_N$ y $a \in \Sigma$, entonces:
  - $q_B \in \delta(q_A, a) \iff A \to aB \in P$
  - $q_f \in \delta(q_A, a) \iff A \to a \in P$
  - $q_A \in F \iff A \to \lambda \in P$
  - $q_f \in F$

Equivalencia clausura transitiva de producciones y δ: Vamos a probar por inducción que

$$A \overset{*}{\Rightarrow} \alpha B \iff q_B \in \hat{\delta}(q_A, \alpha)$$

- **Caso base $\alpha = \lambda$:**
  - $A \overset{*}{\Rightarrow} \alpha B$, pero las gramáticas regulares no acentan producciones que vayan de un no terminal a otro sin pasar por un terminal, por lo que $B = A$. Osea $A \overset{*}{\Rightarrow} \alpha A$.
  - Además, como es un AFND, no tiene transiciones lambda, osea que $\delta(q_A, \lambda) = \{q_A\}$, por lo que $q_A \in \hat{\delta}(q_A, \alpha)$.
- **Caso inductivo $\alpha = \beta a$:**

$$\begin{aligned} A \overset{*}{\Rightarrow} \alpha B &\iff A \overset{*}{\Rightarrow} \beta a B \underset{\text{def.}}{\iff} \exists C \in V_N : A \overset{*}{\Rightarrow} \beta C \wedge C \to aB \\ &\underset{\substack{\text{H.I} \\ \text{constr. M}}}{\iff} \exists q_C \in Q, q_c \in \hat{\delta}(q_A, \alpha) \wedge q_B \in \delta(q_C, a) \\ &\iff q_B \in \delta(\hat{\delta}(q_A, \alpha), a) \\ &\iff q_B \in \hat{\delta}(q_A, \beta a) \iff q_B \in \hat{\delta}(q_A, \alpha) \end{aligned}$$

Demostración de la equivalencia: Vamos a demostrar que el lenguaje generado por G y M son iguales, osea que $\alpha a \in \mathcal{L}(M) \iff S \overset{*}{\Rightarrow} \alpha a$. Como G es una grámatica regular, hay solo dos formas de llegar desde S hasta αa:

1. $\exists A \in V_N : S \overset{*}{\Rightarrow} \alpha A \wedge A \to a \in P$
2. $\exists B \in V_N : S \overset{*}{\Rightarrow} \alpha a B \wedge B \to \lambda \in P$

Entonces:

$$\begin{aligned} S \overset{*}{\Rightarrow} \alpha a &\underset{\text{def. G}}{\iff} (\exists A \in V_N : S \overset{*}{\Rightarrow} \alpha A \wedge A \to a \in P) \vee (\exists B \in V_N : S \overset{*}{\Rightarrow} \alpha a B \wedge B \to \lambda \in P) \\ &\underset{\text{Equiv. anterior}}{\iff} (\exists q_A \in Q, q_A \in \hat{\delta}(q_0, \alpha) \wedge q_f \in \delta(q_A, a)) \vee (\exists q_B \in Q, q_B \in \hat{\delta}(q_0, \alpha a) \wedge q_B \in F) \\ &\underset{\text{def. } \delta}{\iff} q_f \in \delta(q_S, \alpha a) \vee (\exists q_B \in Q, q_B \in \hat{\delta}(q_0, \alpha a) \wedge q_B \in F) \\ &\iff \alpha a \in \mathcal{L}(M) \end{aligned}$$

Falta ver que pasa si $\lambda \in \mathcal{L}(G)$:

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

1.3.4. AFD a gramática regular

Dado un AFD $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, existe una gramática regular $G = \langle V_N, V_T, P, S \rangle$ equivalente

Demonstración

Contrucción de G: Vamos a construir G de la siguiente forma:

- $V_N = Q$, para mayor claridad llamamos A_p al no terminal correspondiente al estado $p \in Q$
- $V_T = \Sigma$
- $S = q_0$
- Si $q \in Q \land q \notin F$ entonces $A_p \to aA_q \in P \iff \delta(p, a) = q$
- Si $q \in F$ entonces $A_p \to a \in P \iff \delta(p, a) = q$
- $S \to \lambda \in P \iff q_0 \in F$

Paso intermedio: Vamos a demostrar por inducción:

$$\hat{\delta}(p, \alpha) = q \iff A_p \overset{*}{\Rightarrow} \alpha A_q$$

- **Caso base:** $\alpha = \lambda$ es trivial:

$$\hat{\delta}(p, \lambda) = q \iff A_p \overset{*}{\Rightarrow} A_p$$

- **Caso inductivo** $\alpha = \beta a$**:** Queremos probar que $\hat{\delta}(p, \alpha) = q \iff A_p \overset{*}{\Rightarrow} \alpha A_q$.
  Nuestra hipotesis inductiva: $\hat{\delta}(p, \beta) = q \iff A_p \overset{*}{\Rightarrow} \beta A_q$ para todo $|\beta| \leq n$

$$\hat{\delta}(p, \alpha) = \hat{\delta}(p, \beta a) = q \underset{\text{def.}}{\iff} \exists r \in Q : \ \hat{\delta}(p, \beta) = r \land \delta(r, a) = q$$
$$\underset{\substack{\text{H.I} \\ \text{constr. G}}}{\iff} \exists A_r, A_p \overset{*}{\Rightarrow} \beta A_r \land A_r \to aA_q \in P \iff A_p \overset{*}{\Rightarrow} \beta a A_q$$

Demostración de equivalencia de lenguajes:

$$\alpha a \in \mathcal{L}(M) \underset{\text{def.}}{\iff} \hat{\delta}(q_0, \alpha a) \in F \underset{\text{def.}}{\iff} \exists q \in Q : \ \hat{\delta}(q_0, \alpha) = q \land \delta(q, a) \in F$$
$$\underset{\text{paso intermedio}}{\iff} \exists A_p, A_{q0} \overset{*}{\Rightarrow} \alpha A_p \land A_p \to a \in P \iff A_{q0} \overset{*}{\Rightarrow} \alpha a$$
$$\underset{\text{def.}}{\iff} \alpha a \in \mathcal{L}(G)$$

1.4. Minimización de AFD

1.4.1. Indistinguibilidad

Sea $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ un AFD, decimos que $p, q \in Q$, son indistinguibles ($p \equiv q$) si para toda cadena $\alpha \in \Sigma^*$ tal que $\hat{\delta}(p, \alpha) \in F$ entonces pasa que $\hat{\delta}(q, \alpha) \in F$ y viceversa. Si $p, q \in Q$ son indistinguibles, entonces decimos que p y q son equivalentes.

$$p \equiv q \iff \forall \alpha \in \Sigma^* : \ (\hat{\delta}(p, \alpha) \in F \iff \hat{\delta}(q, \alpha) \in F)$$

Teorema: Si p y q son indistinguibles, sea $\alpha \in \Sigma^*$ entonces $\hat{\delta}(p, \alpha) \equiv \hat{\delta}(q, \alpha)$

$$p \equiv q \implies \forall \alpha \in \Sigma^* : \ \hat{\delta}(p, \alpha) \equiv \hat{\delta}(q, \alpha)$$

DEMOSTRACIÓN

Sean $p, q \in Q$, $p \equiv q$.
Supogamos que existe $\alpha \in \Sigma^*$ tal que $\hat{\delta}(p, \alpha) \not\equiv \hat{\delta}(q, \alpha)$ entonces existe una cadena $\gamma \in \Sigma^*$ que distingue a $\hat{\delta}(p, \alpha)$ de $\hat{\delta}(q, \alpha)$. Osea que $\hat{\delta}(\hat{\delta}(p, \alpha), \gamma) \in F$ y $\hat{\delta}(\hat{\delta}(q, \alpha), \gamma) \notin F$ (o viceversa).
Por def: $\hat{\delta}(\hat{\delta}(p, \alpha), \gamma) = \hat{\delta}(p, \alpha\gamma)$ y $\hat{\delta}(\hat{\delta}(q, \alpha), \gamma) = \hat{\delta}(p, \alpha\gamma)$. Entonces, como $\alpha\gamma$ es una cadena que nos permite distinguir p de q, es decir $p \not\equiv q$. Absurdo.

Teorema: $\equiv$ es una relación de equivalencia.

DEMOSTRACIÓN

- **Reflexividad:** $p \equiv p$:

$$\forall \alpha \in \Sigma^* : \ (\hat{\delta}(p, \alpha) \in F \iff \hat{\delta}(p, \alpha) \in F) \iff p \equiv p$$

- **Simetría:** $p \equiv q \implies q \equiv p$:

$$\begin{aligned} p \equiv q &\implies \forall \alpha \in \Sigma^* : \ (\hat{\delta}(p, \alpha) \in F \iff \hat{\delta}(q, \alpha) \in F) \\ &\iff \forall \alpha \in \Sigma^* : \ (\hat{\delta}(q, \alpha) \in F \iff \hat{\delta}(p, \alpha) \in F) \iff q \equiv p \end{aligned}$$

- **Transitividad:** $p \equiv q \wedge q \equiv r \implies p \equiv r$:

$$\begin{aligned} p \equiv q &\implies \forall \alpha \in \Sigma^* : \ (\hat{\delta}(p, \alpha) \in F \iff \hat{\delta}(q, \alpha) \in F) \\ q \equiv r &\implies \forall \alpha \in \Sigma^* : \ (\hat{\delta}(q, \alpha) \in F \iff \hat{\delta}(r, \alpha) \in F) \end{aligned}$$

  Entonces

$$\forall \alpha \in \Sigma^* : \ (\hat{\delta}(p, \alpha) \in F \iff \hat{\delta}(r, \alpha) \in F) \iff p \equiv r$$

Indestinguibilidad de orden k

$$p \overset{k}{\equiv} q \iff \forall \alpha \in \Sigma^*, (|\alpha| \leq k) \implies (\hat{\delta}(p,\alpha) \in F \iff \hat{\delta}(q,\alpha) \in F)$$

Propiedades:

1. $\overset{k}{\equiv}$ es un relación de equivalencia.

DEMOSTRACIÓN

Es exactamente igual a la demostración $\equiv$ es transitiva.

2. $\overset{k+1}{\equiv} \subseteq \overset{k}{\equiv}$

DEMOSTRACIÓN

$$p \overset{k+1}{\equiv} q \implies \forall \alpha \in \Sigma^*, (|\alpha| \leq k+1) \implies (\hat{\delta}(p,\alpha) \in F \iff \hat{\delta}(q,\alpha) \in F)$$

Ahora como esto vale $\forall \alpha \in \Sigma^*, (|\alpha| \leq k+1)$, necesariamente vale $\forall \alpha \in \Sigma^*, (|\alpha| \leq k)$. Entonces

$$\forall \alpha \in \Sigma^*, (|\alpha| \leq k+1) \implies (\hat{\delta}(p,\alpha) \in F \iff \hat{\delta}(q,\alpha) \in F) \implies p \overset{k}{\equiv} q$$

3. $\left(Q/\overset{0}{\equiv}\right) = \{Q-F, F\}$ si $Q-F \neq \emptyset$ y $F \neq \emptyset$. En castellano, $\overset{0}{\equiv}$ divide al conjunto de estados en estados finales y no finales.

4. $p \overset{k+1}{\equiv} q \iff \left(p \overset{0}{\equiv} q\right) \wedge \left(\forall a \in \Sigma, \delta(p,a) \overset{k}{\equiv} \delta(q,a)\right)$

DEMOSTRACIÓN

$\Rightarrow$) Como $\overset{k+1}{\equiv} \subseteq \overset{k}{\equiv}$ entonces $p \overset{k+1}{\equiv} q \implies p \overset{0}{\equiv} q$.

Por otro lado, supongamos que no vale $\left(\forall a \in \Sigma, \delta(p,a) \overset{k}{\equiv} \delta(q,a)\right)$ entonces

$$\exists a \in \Sigma,\ \exists \alpha \in \Sigma^*, (|\alpha| \leq k) \wedge \hat{\delta}(\delta(p,a),\alpha) \in F \wedge \hat{\delta}(\delta(q,a),\alpha) \notin F$$

o viceversa. Pero entonces $p \overset{k+1}{\not\equiv} q$ ya que $a\alpha \leq k+1$ y $a\alpha$ distinque a p y a q.

$\Leftarrow$) Supogamos que $p \overset{k}{\equiv} q$. Entonces ó $p \overset{0}{\equiv} q$ ó $\exists a\alpha,\ |a\alpha| \leq k+1$ que distingue p de q, o sea que:

$$\hat{\delta}(\delta(p,a),\alpha) \in F \wedge \hat{\delta}(\delta(q,a),\alpha) \notin F$$

o viceversa. Pero entonce $\delta(p,a) \overset{k+1}{\not\equiv} \delta(q,a)$.

5. $\left(\overset{k+1}{\equiv} = \overset{k}{\equiv}\right) \implies \forall n \geq 0, \left(\overset{k+n}{\equiv} = \overset{k}{\equiv}\right)$

DEMOSTRACIÓN

Lo vamos a demostrar por inducción:

Caso base:$n = 0$. Entonces $k \overset{k}{\equiv} = \overset{k}{\equiv}$.

Paso inductivo: Suponemos que es cierto para n, osea que vale $\overset{k+1}{\equiv} = \overset{k}{\equiv} \implies \overset{k+n}{\equiv} = \overset{k}{\equiv}$

Queremos probar $\overset{k+1}{\equiv} = \overset{k}{\equiv} \implies \overset{k+n+1}{\equiv} = \overset{k}{\equiv}$:

Sabemos que $\overset{k+n+1}{\equiv} = \overset{k}{\equiv}$ si y solo si $\forall p,q \in Q, \left(p \overset{k}{\equiv} q \iff p \overset{k+n+1}{\equiv} q\right)$

Por la propiedad (4), tenemos:

$$\begin{aligned} p \overset{k+n+1}{\equiv} &\iff \left(p \overset{0}{\equiv} q\right) \wedge \left(\forall a \in \Sigma, \delta(p,a) \overset{k+n}{\equiv} \delta(q,a)\right) \left(p \overset{0}{\equiv} q\right) \\ &\wedge \left(\forall a \in \Sigma, \delta(p,a) \overset{k+n}{\equiv} \delta(q,a)\right) \\ &\underset{\text{prop. 2}}{\implies} \left(p \overset{0}{\equiv} q\right) \wedge \left(\forall a \in \Sigma, \delta(p,a) \overset{k}{\equiv} \delta(q,a)\right) \\ &\underset{\text{prop. 4}}{\implies} p \overset{k+1}{\equiv} q \underset{\text{prop. 2}}{\implies} q \overset{k}{p} \end{aligned}$$

1.4.2. Autómatas finito determinístico mínimo

Sea $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ un AFD sin estados inaccesibles, el AFD mínimo equivalente $M' = \langle Q', \Sigma, \delta', q_0, F'\rangle$ se define de la siguiente manera:

- $Q' = Q/\equiv$. Vamos a notar $[q]$ al estado que representa a la clase de equivalencia que contiene a q.
- $\delta'([q],a) = [\delta(q,a)]$
- $q_0' = [q_0]$
- $F' = \{[q] \in Q' \ :\ q \in F\}$

Teorema:

$$\forall \alpha \in \Sigma^*, \hat{\delta}(q, \alpha) = r \implies \hat{\delta}'(q'_0, \alpha) = \hat{\delta}'([q], \alpha) = [r]$$

DEMOSTRACIÓN

Va a ser por inducción en la longitud de α:

- $\alpha = \lambda$: $\hat{\delta}(q, \epsilon) = q$, por def. de $\hat{\delta}$

  $\hat{\delta}'(q'_0, \epsilon) = \hat{\delta}'([q], \epsilon) = [q]$ por def. de $\hat{\delta}'$

  Entonces $\hat{\delta}(q, \lambda) = q \implies \hat{\delta}'([q], \lambda) = [q]$

- Paso inductivo: Sea $\alpha = \beta a$, queremos probar que $\hat{\delta}(q, \alpha) = r \implies \hat{\delta}'([q], \alpha) = [r]$.

  Nuestra hipotesis inductiva es $\forall \beta \in \Sigma^*,\ |\beta| \leq n,\ \hat{\delta}(q, \beta) = p \implies \hat{\delta}'([q], \beta) = [p]$

  Entonces:

  $$\hat{\delta}(q, \alpha) = \hat{\delta}(q, \beta a) = \delta(\hat{\delta}(q, \beta), a) = \hat{\delta}(p, a) = r \underbrace{\implies}_{\text{constr. } \delta'} \delta'([p], a) = [r] \ (1)$$

  Además, por hipotesis inductiva sabemos que $\hat{\delta}'([q], \beta) = [p]$, entonces remplazando en el último término de la ecuación (1) obtenemos:

  $$\delta'([p], \alpha) = \delta'(\hat{\delta}'([q], \beta), a) = \hat{\delta}(q, \beta a) = \hat{\delta}(q, \alpha) = [r]$$

1.4.3. Algoritmo de minimización de un AFD

Require: $M = \langle Q, \Sigma, \delta, q_0, F \rangle$
  $P \leftarrow \{Q - F, F\}$ ▷ $P = \overset{0}{\equiv}$, separamos los estados finales de los no finales
  $stop \leftarrow false$
  while $stop = false$ **do**
    $P' \leftarrow \emptyset$
    for $X \in P$ **do** ▷ Separamos cada clase de equivalencia en las subclases de $\overset{n+1}{\equiv}$
      while $\exists e \in X : \neg \texttt{marked}(e, X)$ **do** ▷ Elegimos un nuevo representante para cada clase
        $X_1 \leftarrow \{e\}$
        $\texttt{marked}(e, X)$
        for $e' \in X : e \neq e'$ **do** ▷ Conseguimos los elementos de esa clase
          if $\neg \texttt{marked}(e', X) \wedge (\forall a \in \Sigma,\ [\delta(e, a)] = [\delta(e', a)])$ **then**
            $X_1 \leftarrow X_1 \cup \{e'\}$
            $\texttt{mark}(e', X)$
          end if

```
            end for
            P' ← P' ∪ {X_1}
        end while
    end for
    if P ≠ P' then
        P ← P'
    else
        stop ← true
    end if
end while
```

Lema: Sean $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ y $M' = \langle Q', \Sigma, \delta', q_0', F'\rangle$ dos AFDs. Si M no poseee estados inaccesibles y todo par de cadenas que conducen a estados diferentes de M conducen a estados diferentes de M', entonces la cantidad de estados de M' es mayor o igual a la cantidad de estados de M. Es decir:

$$\Big(\forall \alpha, \beta \in \Sigma^*,\ \hat{\delta}(q, \alpha) \neq \hat{\delta}(q, \beta) \implies \hat{\delta}'(q_0', \alpha) \neq \hat{\delta}'(q_0', \beta)\Big) \implies |Q| \leq |Q'|$$

DEMOSTRACIÓN

Sea $g : Q \to \Sigma^*$ definida por $g(q) = \text{mín}\Big\{\alpha \in \Sigma^* :\ \hat{\delta}(q_0, \alpha) = q\Big\}$ donde suponemos una relación de orden en Σ^* dada por la longitud para cadenas de distinta longitud, y por el lexicográfico para las cadenas de igual longitud.
Definamos $f : Q \to Q'$ con $f(q) = \hat{\delta}'(q_0', g(q))$.
Como para cualquier par de estados diferentes $p, q \in Q$ es cierto que $\hat{\delta}(q_0, g(p)) \neq \hat{\delta}(q_0, g(q))$, entonces $\hat{\delta}'(q_0', g(p)) \neq \hat{\delta}'(q_0', g(q))$. Lo que equivale a decir que $f(p) \neq f(q)$. Por lo tanto, f es una función inyectiva, es decir que $|Q| \leq |Q'|$.

Lema: Sea $M_R = \langle Q_R, \Sigma, \delta_R, q_{R0}, F_R\rangle$ el autómata reducido correspondiente a $M = \langle Q, \Sigma, \delta, q_0, F\rangle$. Entonces, cualquier autómata $M' = \langle Q', \Sigma, \delta', q_{0'}, F'\rangle$ que reconozca el mismo lenguaje que M no poseerá menos estados que M_R. Osea:

$$\forall M',\ \mathcal{L}(M') = \mathcal{L}(M) \implies |Q'| \geq |Q_R|$$

DEMOSTRACIÓN

Supongamos que $\exists M'$ tal que $|Q'| < |Q_R|$, entonces según el lema anterior deben existir dos cadenas $\alpha, \beta \in \Sigma^*$ tales que

$$\hat{\delta_R}(q_0, \alpha) \neq \hat{\delta_R}(q_0, \beta) \wedge \hat{\delta'}(q'_0, \alpha) = \hat{\delta'}(q'_0, \beta)$$

Pero entonces, como $\hat{\delta_R}(q_0, \alpha)$ y $\hat{\delta_R}(q_0, \beta)$ son estados diferentes, entonces $\hat{\delta}(q_0, \alpha)$ y $\hat{\delta}(q_0, \beta)$ son estados distinguibles por pertenecer al autómata reducido M_R entonces $\exists \gamma \in \Sigma^*$ tal que:

$$\hat{\delta}(q_0, \alpha\gamma) \in F \wedge \hat{\delta}(q_0, \beta\gamma) \notin F$$

o viceversa. Entonces $\alpha\gamma \in \mathcal{L}(M_R) \iff \beta\gamma \notin \mathcal{L}(M_R)$.
Por otro lado, como $\hat{\delta'}(q'_0, \alpha) = \hat{\delta'}(q'_0, \beta)$, es obvio que

$$\hat{\delta'}(q'_0, \alpha\gamma) \in F \wedge \hat{\delta'}(q'_0, \beta\gamma) \in F$$

o ninguno de los dos perteneces a F. De esto se inifiere que $\alpha\gamma \in \mathcal{L}(M') \iff \beta\gamma \in \mathcal{L}(M')$.
Pero entonces, $\mathcal{L}(M') \neq \mathcal{L}(M)$, se contradice nuestra hipotesis inicial.

1.5. Lenguajes regulares

1.5.1. Lema de pumping

Propiedad: Sea $\mathcal{L}$ un lenguaje regular, si las longitudes de las cadenas de un lenguaje $\mathcal{L}$ están acotadas superiormente, entonces $\mathcal{L}$ tiene que ser finito.

Propiedad: Si $\mathcal{L}$ es un luenguaje regular infinito, entonces el grafo de un autómata fínito que acepte $\mathcal{L}$ tiene que tener un camino desde el estado inicial hasta algún estado final que paso por algún ciclo.

Lema de pumping: Sea $\mathcal{L}$ un lenguaje regular, si $\mathcal{L}$ es infinito, entonces todas las cadenas ω de longitud mayor o igual a n (para algún $n > 1$) van a ser de la forma $\omega = xy^iz$, es decir hay una parte de ω que se repite i cantidad de veces:

$$\mathcal{L} \text{ es regular e infinito } \implies \exists n \in \mathbb{N} \text{ tal que}$$
$$\forall \omega \in \mathcal{L},\ |\omega| \geq n : \left(\exists x, y, z \in \Sigma^* : \omega = xyz \wedge |xy| \leq n \wedge |z| \geq 1 \wedge (\forall i \geq 0 :\ xy^iz \in \mathcal{L}\right)$$

DEMOSTRACIÓN

Supongamos que $\mathcal{L}$ es un lenguaje regular. Entonces existe una AFD $A = \langle Q, \Sigma, \delta, q_0, F\rangle$ que acepta $\mathcal{L}$.
Sea $n = |Q|$ la cantidad de estados de A y $\omega = a_1a_2 \dots a_m \in \Sigma^*$ de longitud $m > n$. Para cada $i = 0, \dots, m$ definamos el estado $p_i = \hat{\delta}(q_0, a_1 \dots a_i)$ (el estado en el que se encuentra A después de haber consumido los primeros i símbolos de ω.
Como el A solo tiene n estados pero hay $m > n$ estados p_i, es imposible que todos sean distintos. Por lo tanto, existen $0 \leq i < j \leq n$ tales que $p_i = p_j$.
Considerar entonces la siguiente descomposición para $\omega = xyz$:

$x = a_1 \dots a_i$

$y = a_{i+1} \dots a_j$

$z = a_{j+1} \dots a_m$

Entonces podemos concluir que:

$\hat{\delta}(q_0, x) = p_i$

$\hat{\delta}(p_i, y) = p_j$ (que como son el mismo estado implica que A tiene un ciclo)

$\hat{\delta}(p_j, z) = p_m$

Observar que x podría ser la cadena vacía cuando $i = 0$, z podría ser vacía si $j = n = m$, pero y no puede ser vacía ya que se tomó $i < j$.
Vimos entonces que si $\mathcal{L}$ es regular y $|\omega| \geq n$ entonces podemos dividirla en cadenas x, y, z tal que $|xy| \leq n$ y $|z| \geq 1$. Ahora vamos a ver que si $xyz \in \mathcal{L}$ entonces $xy^k z \in \mathcal{L}$ para todo $k \geq 0$.

- Si $k = 0$ entonces $xy^0 z = xz$:

  $$\hat{\delta}(q_0, xz) = \hat{\delta}(\hat{\delta}(q_0, x), z) = \hat{\delta}(p_i, z) = \hat{\delta}(p_j, z) = p_m$$

  Y p_m es un estado final pues es el mismo estado al que llegamos si la entrada fuese $xyz \in \mathcal{L}$. Entonces $xz = xy^0 z \in \mathcal{L}$.

- Si $i > 0$. Entonces A consume x desde q_0 y llega hasta p_i. Luego A consume y desde p_i y llega hasta p_j (que son iguales) y repite este ciclo k veces hasta consumir todas las apariciones de y en la cadena. Finalmente A consume z desde p_j y llega hasta p_m. Entonces A llega a un estado final y por lo tanto $xy^k z \in \mathcal{L}$.

Contrarecíproco:

$\forall n \in \mathbb{N}\ \exists \omega \in \mathcal{L}$ tal que $|\omega| \geq n \wedge \forall x, y, z \in \Sigma^* : \omega \neq xyz \vee |xy| \geq n \vee |z| \leq 1 \vee \exists i \geq 0,\ xy^i z \notin \mathcal{L}$
$\implies \mathcal{L}$ no es regular

1.5.2. Operaciones sobre lenguajes regulares

Unión de lenguajes regulares

Dados M_1 y M_2 AFDs que aceptan los lenguajes L_1 y L_2, respectivamente, podemos construir un AFD $M = \langle Q_\cup, \Sigma, \delta_\cup, q_{0\cup}, F_\cup \rangle$ que acepte el lenguaje $L_1 \cup L_2$ con:

- $Q_\cup = Q_1 \times Q_2$
- $\delta_\cup((q, r), a) = (\delta_1(q, a), \delta_2(r, a))$ para $q \in Q_1$, $r \in Q_2$ y $a \in \Sigma$
- $q_{0\cup} = (q_{0_1}, q_{0_2})$
- $F_\cup = \{(q, r) \in Q_\cup \mid q \in F_1 \vee r \in F_2\}$

DEMOSTRACIÓN

$$\begin{aligned}
\alpha \in \mathcal{L}(M_\cup) &\iff \hat{\delta}_\cup((q_{0_1}, q_{0_2}), \alpha) \in F_\cup \iff (\hat{\delta}_1(q_{0_1}, \alpha), \hat{\delta}_2(q_{0_2}, \alpha)) \in F_\cup \\
&\iff \hat{\delta}_1(q_{0_1}, \alpha) \in F_1 \vee \hat{\delta}_2(q_{0_2}, \alpha) \in F_2 \\
&\iff \alpha \in \mathcal{L}(M_1) \vee \alpha \in \mathcal{L}(M_2)
\end{aligned}$$

Intersección de lenguajes regulares

Dados M_1 y M_2 AFDs que aceptan los lenguajes L_1 y L_2, respectivamente, podemos construir un AFD $M_\cap = \langle Q_\cap, \Sigma, \delta_\cap, q_{0\cap}, F_\cap \rangle$ que acepte el lenguaje $L_1 \cap L_2$ con:

- $Q_\cap = Q_1 \times Q_2$
- $\delta_\cap((q,r),a) = (\delta_1(q,a), \delta_2(r,a))$ para $q \in Q_1$, $r \in Q_2$ y $a \in \Sigma$
- $q_{0\cap} = (q_{0_1}, q_{0_2})$
- $F_\cap = \{(q,r) \in Q_\cap \mid q \in F_1 \wedge r \in F_2\}$

DEMOSTRACIÓN

$$\begin{aligned}\alpha \in \mathcal{L}(M_\cap) &\iff \hat{\delta}_\cap((q_{0_1}, q_{0_2}), \alpha) \in F_\cap \iff (\hat{\delta}_1(q_{0_1}, \alpha), \hat{\delta}_2(q_{0_2}, \alpha)) \in F_\cap \\ &\iff \hat{\delta}_1(q_{0_1}, \alpha) \in F_1 \wedge \hat{\delta}_2(q_{0_2}, \alpha) \in F_2 \\ &\iff \alpha \in \mathcal{L}(M_1) \wedge \alpha \in \mathcal{L}(M_2)\end{aligned}$$

Complemento de lenguajes regulares

Teorema 1.5.1. El conjunto de lenguajes regulares incluido en Σ^* es cerrado respecto de la complementación. Es decir, si $L \in \mathcal{L}(\Sigma^*)$ es un lenguaje regular entonces $\bar{L} \in \mathcal{L}(\Sigma^*)$ también lo es.

DEMOSTRACIÓN

Sea $\mathcal{L} = \mathcal{L}(M)$ con $M = (Q, \Sigma, \delta, q_0, F)$ un AFD cuya función de transición δ está definida para todos los elementos del alfabeto Σ. Si hay algún símbolo $a \in \Sigma$ que falta en las posibles transiciones desde un estado $q \in Q$ entonces se puede agregar un estado trampa q_{trampa} (no final) y una transición $\delta(q,a) = q_{trampa}$ para completar la definición de la función.
Si armamos el automáta $M_\neg = \langle Q, \Sigma, \delta, q_0, Q - F \rangle$, entnonces $\alpha \in \mathcal{L}M_\neg \iff \delta(q_0, w) \in Q - F \iff \alpha \in \Sigma^* - L$.

Teorema 1.5.2. El conjunto de lenguajes regulares es cerrado respecto de la intersección

DEMOSTRACIÓN

Como ya probamos que el conjunto de lenguajes regulares es cerrado respecto de la unión y el complemento entonces si logramos escribir la interesección como combinación de estas dos operaciones podremos decir que también es cerrada respecto de la intersección:

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

Entonces si L_1 y L_2 son lenguajes regulares, entonces sus complementos también lo son y la unión de los complementos y el complemento de la misma también lo son.

Teorema 1.5.3. De estos ultimos tres teoremas puede deducirse que la unión finita y la intersección finita de lenguajes regulares dan por resultado un lenguaje regular.

$$\forall n \in \mathbb{N},\ \bigcup_{i=1}^{n} L_i \text{ es regular}$$

$$\forall n \in \mathbb{N},\ \bigcap_{i=1}^{n} L_i \text{ es regular}$$

DEMOSTRACIÓN

Por inducción sobre n:

Caso base: $n = 0$.

$$\bigcup_{i=1}^{0} L_i = \emptyset \text{ es regular}$$

$$\bigcap_{i=1}^{0} L_i = \emptyset \text{ es regular}$$

- **Caso inductivo:** $n \geq 1$. Supongamos que $L_1, \ldots, L_n$ son lenguajes regulares y que $\bigcup_{i=1}^{n-1} L_i$ y $\bigcap_{i=1}^{n-1} L_i$ son lenguajes regulares. Entonces:

$$\bigcup_{i=1}^{n} L_i = \bigcup_{i=1}^{n-1} L_i \cup L_n \text{ es regular}$$

$$\bigcap_{i=1}^{n} L_i = \bigcap_{i=1}^{n-1} L_i \cap L_n \text{ es regular}$$

Teorema 1.5.4. Todo lenguaje finito es regular

DEMOSTRACIÓN

Sea L un lenguaje finito tal que $|L| = n$ y $x_i \in L$ con $1 \leq i \leq n$.
Entonces podemos definir n lenguajes $L_i = \{x_i\}$ regulares. Por el teorema anterior, la unión finita de lenguajes regulares es un lenguaje regular. Entonces $L = \bigcup_{i=1}^{n} L_i$ es un lenguaje regular.

1.5.3. Problemas decicibles acerca de lenguajes regulares

1. **Pertenencia:** Dado un lenguaje regular L y una cadena w, determinar si $w \in L$.

   Se construye un AFD M que reconozca L y se verifica si w es aceptada por M.

2. **Finitud:** Dado un lenguaje regular L, determinar si L es finito.

   Un lenguaje regular L es finito, si en su AFD ningún ciclo que es alcanzable desde el estado inicial puede, a su ve, alcanzar algún estado final.

   $$L \text{ finito } \iff \Big(\forall q \in Q, \Big(q_0 \overset{*}{\Rightarrow} q \wedge q \overset{*}{\Rightarrow} f \in F \implies \Big(\nexists q \overset{+}{\rightarrow} q\Big)\Big)\Big)$$

   Escrito en función de δ:

   $$L \text{ finito si y solo si: } \forall q \in Q \text{ tal que } \Big(\exists \alpha, \omega \in \Sigma^*,\ \hat{\delta}(q_0, \alpha) = q \wedge \hat{\delta}(q, \omega) = f \in F\Big)$$
   $$\text{vale que } \Big(\nexists \beta \in \Sigma^+,\ \hat{\delta}(q, \beta) = q\Big)$$

3. **Vacuidad:** Dado un lenguaje regular L, determinar si L es vacío.

   Se construye el AFD M que reconozca L y se verifica si el conjunto A de estados alcanzables de M tiene un estado final. Si $F \cap A = \emptyset$, entonces L es vacío.

4. **Equivalencia:** Dados dos lenguajes regulares L_1 y L_2, determinar si L_1 y L_2 son equivalentes.

   Si el lenguaje $(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$ es vacío, entonces L_1 y L_2 son equivalentes, si no no lo son.

1.6. Autómatas de Pila

Los autómatas de pila (AP) son autómatas fínitos que además tienen la capacidad de almacenar información en una pila con dos operaciones básicas: apilar y desapilar. La operación apilar consiste en agregar un elemento a la pila, mientras que la operación desapilar consiste en eliminar el elemento que se encuentra en la cima de la misma.

Cuando se va a realiar una transición, se consume un elemento de la cadena de entrada y un elemento del tope de la pila. Al moverse al nuevo estado, el automáta pushea (apila) una cadena de símbolos en la pila (que puede ser vacía).

Definición: Un autómata de pila (AP) es una 7-tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ donde:

- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada y es finito.
- Γ es un alfabeto de la pila y es finito.
- δ es la función de transición, que es un conjunto de reglas de la forma:

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$$

$$\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_n, \gamma_n)\}$$

  δ indica el estado que p_i al que se transiciona y la cadena γ_i que se apila en la pila.
- $q_0 \in Q$ es el estado inicial.
- $Z_0 \in \Gamma$ es el símbolo inicial de la pila.
- $F \subseteq Q$ es el conjunto de estados finales.

1.6.1. Configuración instantanea

Una configuración instantánea de un autómata de pila es una 3-tupla $(q, w, \gamma) \in Q \times \Sigma \times \Gamma^*$ donde:

- $q \in Q$ es el estado actual.
- $w \in \Sigma^*$ es la parte de la cadena de entrada que no ha sido consumida.
- $\gamma \in \Gamma^*$ es el contenido de la pila.

La configuracion inicial de un automáta será entonces $\sigma = (q_0, \alpha, Z_0)$ si α es la cadena que usamos de entrada.

Cambio de configuración: Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ un automáta de pila, $a \in \Sigma$, $\alpha \in \Sigma^*$, $t \in \Gamma$, $\tau, \pi \in \Sigma^*$ y $q, r \in Q$. Definimos las transiciones ($\vdash$) entre dos configuraciones instantaneas de la siguiente manera:

- $(q, a\alpha, t\pi) \vdash (r, \alpha, \tau\pi)$ si $(r, \tau) \in \delta(q, a, t)$.
- $(q, \alpha, t\pi) \vdash (r, \alpha, \tau\pi)$ si $(r, \tau) \in \delta(q, \lambda, t)$.

1.6.2. Lenguajes reconocidos por un autómata

Lenguaje aceptado por estado final

Son todas las cadenas $\alpha \in \Sigma^*$ que hacen que el automáta llegue a un estado final $q \in F$.

$$\mathcal{L}(M) = \{\alpha \in \Sigma^* \mid \exists q \in F \text{ tal que } (q_0, \alpha, Z_0) \overset{*}{\vdash} (q, \lambda, Z_0)\}$$

Lenguaje aceptado por pila vacía

Son todas las cadenas $\alpha \in \Sigma^*$ que hacen que el automáta llegue a un estado en el cual la pila quede vacía.

$$\mathcal{L}_\lambda(M) = \{\alpha \in \Sigma^* \mid \exists q \in F \text{ tal que } (q_0, \alpha, Z_0) \overset{*}{\vdash} (q, \lambda, \lambda)\}$$

Teorema 1.6.1. Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ un automáta que acepta el lenguaje $\mathcal{L}(M)$ por estado final, entonces existe $M' = \langle Q', \Sigma, \Gamma', \delta', q_0', X_0, \emptyset\rangle$ tal que $\mathcal{L}_\lambda(M') = \mathcal{L}(M)$.

DEMOSTRACIÓN

Construimos M' de la siguiente manera:

- $Q' = Q \cup \{q_\lambda, q_0'\}$
- $\Gamma' = \Gamma \cup \{X_0\}$
- $\delta' : Q' \times \Sigma \times \Gamma' \to \mathcal{P}(Q' \times \Gamma')$
  1. $\delta'(q_0', \lambda, X_0) = \{(q_0, Z_0X_0)\}$

     M' entra al estado inicial de M con la pila Z_0X_0. como X_0 no es un símbolo de la pila de M, M no puede vaciar la pila.
  2. $\forall q \in Q, a \in \Sigma \cup \{\lambda\}, \delta'(q, a, Z) = \{(r, \tau) \mid (r, \tau) \in \delta(q, a, Z) \text{ con } r \in Q \wedge \tau \in \Gamma^*\}$

     M' sigue las mismas transiciones que M para todos los estados originales.

3. $\forall q \in F, \forall Z \in \Gamma', (q_\lambda, \lambda) \in \delta'(q, \lambda, Z)$

   Cuando M' llega a un estado que es final en M automáticamente se activa el el estado q_λ que desencola el último elemento de la pila (y no pushea nada).

4. $\forall Z \in \Gamma', (q_\lambda, \lambda) \in \delta'(q_\lambda, \lambda, Z)$

   Cuando M' llega al estado q_λ, sin importar cual es el tope de la pila lo desencola hasta que la pila queda vacía (hay un loop sobre q_λ que consume toda la pila).

Veamos que $\mathcal{L}_\lambda(M) = L$, lo vamos a demostrar en dos pasos:

$\mathcal{L}(M) \subseteq \mathcal{L}_\lambda(M)$:

Si $\alpha \in \mathcal{L}(M)$ entonces $(q_0, \alpha, Z_0) \overset{*}{\underset{M}{\vdash}} (q, \lambda, \gamma)$ con $q \in F$ (por def.).

Por la regla 1, tenemos que $(q_0', \alpha, X_0) \overset{*}{\underset{M'}{\vdash}} (q_0, \alpha, Z_0 X_0)$

Por la regla 2, $(q_0, \alpha, Z_0) \overset{*}{\underset{M'}{\vdash}} (q, \lambda, \gamma)$, entonces

$$(q_0', \alpha, X_0) \overset{*}{\underset{M'}{\vdash}} (q_0, \alpha, Z_0 X_0) \overset{*}{\underset{M'}{\vdash}} (q, \lambda, \gamma X_0)$$

Por las reglas 3 y 4 es cierto que $(q, \lambda, \gamma X_0) \overset{*}{\underset{M'}{\vdash}} (q_\lambda, \lambda, \lambda)$

Por lo tanto, $\alpha \in \mathcal{L}_\lambda(M)$

$\mathcal{L}_\lambda(M) \subseteq L$:

Si $\alpha \in \mathcal{L}_\lambda(M)$ entonces, por como construimos M' vale que

$$(q_0', \alpha, X_0) \underset{M'}{\vdash} \underbrace{(q_0, \alpha, Z_0 X_0) \overset{*}{\underset{M'}{\vdash}} (q, \lambda, \gamma X_0)}_{A} \overset{*}{\underset{M'}{\vdash}} (q_\lambda, \lambda, \lambda)$$

De A se puede ver que $(q_0, \alpha, Z_0) \overset{*}{\underset{M}{\vdash}} (q, \lambda, \gamma)$. Además como los únicos estados que pueden llegar a q_λ son estados finales de M (por 3) vale que $q \in F$. Luego $\alpha \in \mathcal{L}(M)$

Teorema 1.6.2. Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset \rangle$ y $\mathcal{L}_\lambda(M)$ el lenguaje que acepta por pila vacía. Entonces existe $M' = \langle Q', \Sigma, \Gamma', \delta', q_0', X_0, F \rangle$ tal que $\mathcal{L}(M') = \mathcal{L}_\lambda(M)$.

DEMOSTRACIÓN

Construimos M' de la siguiente manera:

- $Q' = Q \cup \{q'_0, q'_f\}$
- $\Gamma' = \Gamma \cup \{X_0\}$
- $F = \{q'_f\}$
- $\delta' : Q' \times (\Sigma \cup \{\lambda\}) \times \Gamma' \to \mathcal{P}(Q' \times \Gamma'^*)$

  1. $\delta'(q'_0, \lambda, X_0) = \{(q_0, Z_0 X_0)\}$

     M' activa automaticamente el estado q_0 con pila $Z_0 X_0$ y como X_0 no es un símbolo de Γ, M no puede consumirlo y vaciar la pila.

  2. $\forall q \in Q, \forall a \in \Sigma \cup \{\lambda\}, \delta'(q, a, Z) = \delta(q, a, Z)$

     Simulación de M.

  3. $\forall q \in Q,\ (q'_f, \lambda) \in \delta(q, \lambda, X_0)$

     Todos los estados a los que se puede llega con la pila vacía en M, llegarán con X_0 en M'. En este caso, podrán saltar automaticamente al estado $q'_f \in F$.

La demostración de que ambos lenguajes son equivalentes es similar a la anterior.

1.6.3. Gramáticas independientes de contexto

Para toda grámatica independiente de contexto G, puede definirse un autómata de pila M que acepta por pila vacía el lenguaje generado por dicha gramática.

DEMOSTRACIÓN

Esto no es una demostración completa.

Sea $G = \langle V_N, V_T, P, S\rangle$ una gramática independiente de contexto, vamos a contruir $M = \langle Q, \Sigma, \Gamma, \delta, q_0, S, \emptyset\rangle$ tal que $\mathcal{L}(M) = \mathcal{L}(G)$:

- $Q = \{q_0\}$
- $\Sigma = V_T$
- $\Gamma = V_N \cup V_T$
- $\delta(q_0, a, t) = \begin{cases} \{(q_0, \alpha) : t \to \alpha \in P\} & \text{si } t \in V_N \wedge a = \lambda \\ \{(q_0, \lambda)\} & \text{si } t \in V_T \wedge a = t \neq \lambda \end{cases}$

Veamos que M acepta $\mathcal{L}(G)$:

- Si en el tope de la pila hay un símbolo no términal $t \in V_N$, entonces el autómata lo remplazará por el lado derecho α de alguna producción que tenga a t en el lado izquierdo. Esto lo hará de manera tal que el símbolo que está más a la izquierda en el lado derecho de dicha producción sea el siguiente tope de la pila.
- Si en el tope de la pila hay un símbolo terminal $t \in V_T$, entonces el autómata verificará que este sea igual al próximo símbolo en la cadena de entrada y lo desapilará.

1.6.4. Autómatas de pila deterministicos

Es un autómata de pila $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ que cumple que para todo $q \in Q,\ a \in \sigma$ y $A \in \Gamma$:

1. $|\delta(q, a, A)| \leq 1$
2. $|\delta(q, \lambda, A)| \leq 1$
3. $|\delta(q, \lambda, A) = 1 \implies |\delta(q, a, A)| = 0$

No es cierto que para cada autómata de pila no deterministico exista un autómata de pila deterministico que acepte el mismo lenguaje.

Propiedad del prefijo

Se dice que un lenguaje L posee la **propiedad del prefijo** si y sollo si para todo par de cadenas **no nulas** x e y es cierto que $x \in L \implies xy \notin L$.

Si un lenguaje L no posee la propiedad del prefijo, entonces todo autómata de pila M que acepte L por pila vacía es necesariamente **no deterministico**

1.7. Gramáticas Independientes del Contexto

Una gramática $G = \langle V_N, V_T, P, S\rangle$ es independiente del contexto si y solo si las producciones en P son de la forma $A \to \alpha$ con $A \in V_N$ y $\alpha \in (V_N \cup V_T)^*$.

Si en particular para toda $A \to \alpha \in P$ pasa que $\alpha \in (V_N \cup V_T)^+$ (o sea, sin reglas borradoras), entonces decimos que G es una **gramática propia**.

1.7.1. Árboles de Derivación

Sea $G = \langle V_N, V_T, P, S\rangle$ una gramática independiente del contexto. Un árbol de derivación de una cadena ω en G es una estructura de árbol tal que:

- Cada vértice posee una etiqueta que pertenece al conjunto $V_N \cup V_T \cup \{\lambda\}$.
- La raíz del árbol posee la etiqueta S.
- Si un vértice es interior, entonces su etiqueta pertenece al conjunto V_N.
- Si un vértice v_n posee la etiqueta A y sus hijos $v_1, \ldots, v_m$ poseen las etiquetas $X_1, \ldots, X_m$, entonces $A \to X_1 \ldots X_m \in P$.
- Si un vértice es hoja, entonces su etiqueta pertenece al conjunto $V_T \cup \{\lambda\}$.
- Si una hoja posee la etiqueta λ, entonces es el único hijo de su.

Camino: Sea $A \in V_N$ y $X \in V_n \cup V_T$ un vértice de un árbol de derivación. El camino de A a X es la secuencia de vértices $A = v_1, v_2, \ldots, v_n = X$ tal que v_i es hijo de v_{i+1} para todo $i \in \{1, \ldots, n-1\}$.

Altura de un árbol: La altura de un árbol de derivación es la longitud del camino más largo de la raíz a una hoja.

Longitud de un camino: La longitud de un camino es la cantidad de arcos que lo componen.

Lema 1.7.1. Sea $G = \langle V_N, V_T, P, S\rangle$ una gramática independiente del contexto con $P \neq \emptyset$ y sea $T(S)$ el árbol de derivación de S en G para algún $\alpha \in \Sigma^*$ de altura h.

$$\text{Si } a = \text{máx}\{|\beta| : A \to \beta \in P\}, \text{ entonces } |\alpha| \leq a^h$$

DEMOSTRACIÓN

Por inducción en h:

- **Caso base:** $h = 0$. El único árbol de derivación posible de esta altura es el símbolo S. Por lo tanto $a^h = a^0 = 1 = |S|$.
- **Paso inductivo:** Sea $\mathcal{T}(S)$ el árbol de derivación para α de altura $h+1$. Sea $\gamma \in (V_N \cup V_T)^+$ una cadena tal que $\gamma \Rightarrow \alpha$ en G. Entonces el árbol de derivación de γ en G tiene altura h.

Por hipotesis inductiva, vale que $|\gamma| \leq a^h$. Por lo tanto, $|\alpha| \leq a^h$.

Por otra parte, $|\alpha| \leq a|\gamma|$ ya que en el peor de los casos cada símbolo de γ usa la reducción de mayor tamaño. Luego, $|\alpha| \leq aa^h = a^{h+1}$.

1.7.2. Gramáticas Ambiguas

Una gramática independiente del contexto G es **ambigua** si y solo si $\exists \alpha \in \mathcal{L}(G)$ tal que posea más de un árbol de derivación.

Lenguaje intrisícamente ambiguo: Un lenguaje independiente del contexto L es **íntrisicamente ambiguo**si y solo si todas las grámaticas que lo tienen como lenguaje son ambiguas.

Derivación más a la izquierda: Una **derivación más a la izquierda** $\left(\underset{L}{\Rightarrow}\right)$ de una cadena ω es aquella que se obtiene remplazando el primer símbolo no terminal que contenga por alguna de sus derivaciónes: Si $A \in V_N$, $\alpha \in V_T^*$, $\beta \in (V_N \cup V_T)^*$ y $A \to \gamma \in P$ entonces $\alpha A \beta \underset{L}{\Rightarrow} \alpha\gamma\beta$

Derivación más a la derecha: Una **derivación más a la derecha** $\left(\underset{R}{\Rightarrow}\right)$ de una cadena ω es aquella que se obtiene remplazando el último símbolo no terminal que contenga por alguna de sus derivaciónes: Si $A \in V_N$, $\alpha \in (V_N \cup V_T)^*$, $\beta \in V_T^*$ y $A \to \gamma \in P$ entonces $\alpha A \beta \underset{R}{\Rightarrow} \alpha\gamma\beta$

1.7.3. Lema de Pumping para Lenguajes Independientes del Contexto

Sea L un lenguaje independiente del contexto sobre un alfabeto Σ, existe $n > 0$ tal que para todo $\alpha \in L$, $|\alpha| \geq n$ existe $r, x, y, z, s \in \Sigma^*$ tales que:

1. $\alpha = rxyzs$
2. $|xyz| \leq n$
3. $|xz| > 0$
4. $\forall i \geq 0, rx^iyz^is \in L$

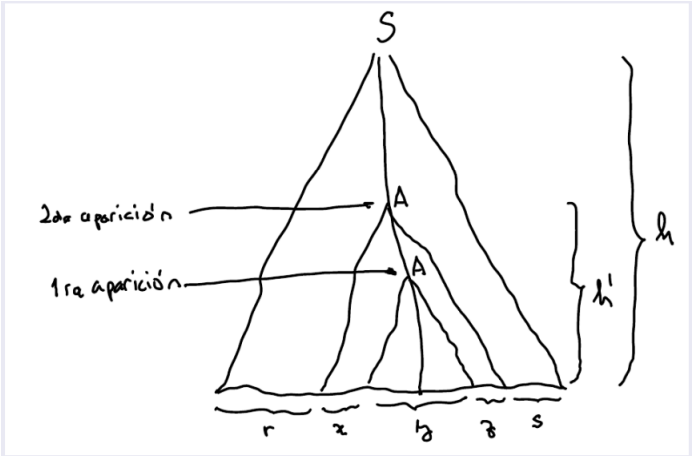
Demostración

Sea $G = \langle V_N, V_T, P, S\rangle$ una gramática independiente del contexto y sea $a = \text{máx}\{|\beta| : A \to \beta \in P\}$.

1. Tomemos $n = a^{|V_N|+1}$. Sea $\alpha \in \mathcal{L}(G)$ tal que $|\alpha| \geq n$ y sea $T(S)$ un árbol mínimo de derivación de α en G, es decir tiene la mínima altura posible y tal que no existe otro con menos derivaciones.

   Por el lema 1.7.1, resulta que $a^h \geq |\alpha| \geq n = a^{|V_N|+1}$. Por lo tanto, $h \geq |V_N| + 1$. Entonces existe algún símbolo $b \in \alpha$ tal que su camino desde la raíz es de longitud $h \geq |V_N| + 1$.

Como la cantidad de símbolos no terminales es $|V_N|$, entonces en ese camino seguramente existe un no-terminal A repetido. Recorriendo el camino de forma ascendente, buscamos sus primeras dos apariciones:

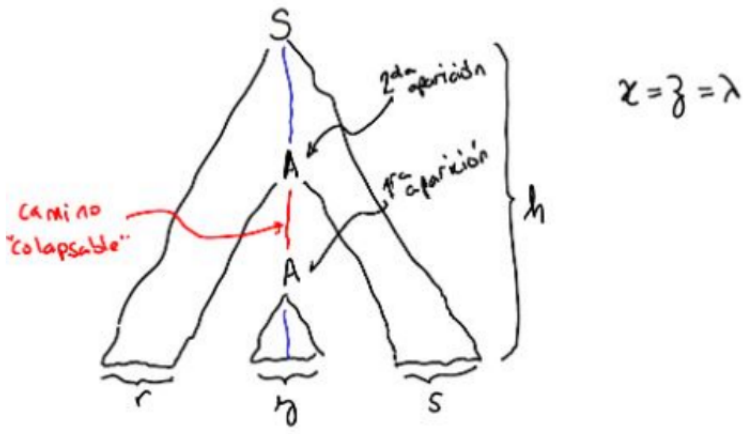


Entonces podemos escribir $\alpha = rxyzs$ con r, x, y, z y s como se muestra en la figura.

2. $|xyz| \leq n = a^{|V_n|+1}$. Como A es el primer no terminal que se repite, podemos asegurar que $h' \leq |V_N| + 1$. La segunda aparición de A tiene que pasar a lo sumo $|V_n|$ pasos mas adelantes, sino se debería repetir otro no terminal antes. Entonces por el lema 1.7.1, resulta que $|xyz| \leq a^{h'} \leq a^{|V_N|+1}$.

3. $|xz| > 0$.

   Supongamos que $|xz| = 0$, esto quiere decir que podriamos remplazar el subárbol con raíz en la segunda aparición de A por el subárbol con raíz en la primera aparición.



Esto es absurdo ya que habíamos dicho que el árbol era mínimo por lo que no debería poder colapsarse ninguno de sus caminos.

4. $\forall i \geq 0, rx^iyz^is \in L$ Finalmente, vamos a demostrar esto por inducción:

   - **Caso base ($i = 0$):** Sabemos que $S \overset{*}{\Rightarrow} rAs$ y $A \overset{*}{\Rightarrow} y$, entonces $S \overset{*}{\Rightarrow} rAs \overset{*}{\Rightarrow} rys$. Además $rys = rx^0yz^0s$.
   - **Paso inductivo $i-1 \Rightarrow i$:**

     Hipotesis inductiva: $S \overset{*}{\Rightarrow} rx^{i-1}Az^{i-1}s$.

     Sabemos $A \overset{*}{\Rightarrow} xyz$, entonces:

     $$S \overset{*}{\Rightarrow} rx^{i-1}Az^{i-1}s \Rightarrow rx^{i-1}xyzz^{i-1}s \Rightarrow rx^iyz^is$$

     Y por lo tanto $rx^iyz^is \in L$.

1.7.4. Propiedades de los Lenguajes Independientes del Contexto

- Si L_1 y L_2 son lenguajes independientes del contexto, entonces $L_1 \cup L_2$ también lo es.

DEMOSTRACIÓN

Como L_1 y L_2 son dos lenguajes independientes del contexto, entonces existen dos grámatica independientes del contexto $G_1 = \langle V_N, \Sigma, P_1, S_1\rangle$ y $G_2 = \langle V_N, \Sigma, P_2, S_2\rangle$. tales que $L_1 = \mathcal{L}(G_1)$ y $L_2 = \mathcal{L}(G_2)$. Supongamos, además, sin perdida de generlidad que $V_{N_1} \cap V_{N_2} = \emptyset$. Definamos entonces:

$$G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1,\ S \to S_2\}, S\rangle$$

Veamos que $\forall \alpha \in \Sigma^*, \alpha \in \mathcal{L}(G) \iff \alpha \in \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$.

$$\begin{aligned} \alpha \in \mathcal{L}(G) &\iff S \overset{*}{\Rightarrow} \alpha \iff S_1 \overset{*}{\Rightarrow} \alpha \vee S_2 \overset{*}{\Rightarrow} \alpha \\ &\iff \alpha \in \mathcal{L}(G_1) \vee \alpha \in \mathcal{L}(G_2) \\ &\iff \alpha \in \mathcal{L}(G_1) \cup \mathcal{L}(G_2) \end{aligned}$$

- Si L_1 y L_2 son lenguajes independientes del contexto, entonces L_1L_2 también lo es.

DEMOSTRACIÓN

Como L_1 y L_2 son dos lenguajes independientes del contexto, entonces existen dos gramáticas independientes de contexto $G_1 = \langle V_{N_1}, \Sigma, P_1, S_1\rangle$ y $G_2 = \langle V_{N_2}, \Sigma, P_2, S_2\rangle$ tales que $L_1 = \mathcal{L}(G_1)$ y $L_2 = \mathcal{L}(G_2)$. Supongamos, además, sin perdida de generlidad que $V_{N_1} \cap V_{N_2} = \emptyset$. Definamos entonces:

$$G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S\rangle$$

Veamos que $\forall \alpha \in \Sigma^*, \alpha \in \mathcal{L}(G) \iff \alpha \in \mathcal{L}(G_1)\mathcal{L}(G_2)$.

$$\begin{aligned}
\alpha \in \mathcal{L}(G) &\iff S \overset{*}{\Rightarrow} \alpha \iff S_1 S_2 \overset{*}{\Rightarrow} \alpha \\
&\iff \exists \beta_1, \beta_2 \in \Sigma^* : \ \alpha = \beta_1\beta_2 \wedge S_1 \overset{*}{\Rightarrow} \beta_1 \wedge S_2 \overset{*}{\Rightarrow} \beta_2 \\
&\iff \beta_1 \in \mathcal{L}(G_1) \wedge \beta_2 \in \mathcal{L}(G_2) \\
&\iff \beta_1\beta_2 = \alpha \in \mathcal{L}(G_1)\mathcal{L}(G_2)
\end{aligned}$$

- Si L es un lenguaje independiente del contexto, entonces L^+ también lo es.

DEMOSTRACIÓN

Como L es un lenguaje independiente del contexto, entonces existe una gramática independiente del contexto $G = \langle V_N, \Sigma, P, S\rangle$ tal que $L = \mathcal{L}(G)$. Definamos entonces:

$$G^+ = \langle V_N \cup \{S\}, \Sigma, P \cup \{S' \to SS',\ S' \to S\}, S'\rangle$$

Veamos que $\mathcal{L}(G') = L^+$

$$\alpha \in \mathcal{L}(G') \iff S' \overset{*}{\Rightarrow} \alpha \iff SS' \overset{*}{\Rightarrow} \alpha \vee S \overset{*}{\Rightarrow} \alpha$$

Supongamos que $S \overset{*}{\Rightarrow} \alpha$, entonces trivialmente vale $\alpha \in L \subseteq L^+$.
Por inducción, Supongamos que $S' \overset{*}{\Rightarrow} \beta \in L^+$: Sea $\alpha = \gamma\beta$ con $\gamma \in \Sigma^*$. Entonces $SS' \overset{*}{\Rightarrow} \alpha \iff SS' \overset{*}{\Rightarrow} \gamma\beta \iff S \overset{*}{\Rightarrow} \gamma \wedge S' \overset{*}{\Rightarrow} \beta$ Entonces pasa que $\gamma \in L$ y, por hipotesis inducción,$\beta \in L^+$, Luego $\gamma\beta \in L^+$

- Si L es un lenguaje independiente del contexto, entonces L^* también lo es.

DEMOSTRACIÓN

Como L es un lenguaje independiente del contexto, entonces existe una gramática independiente del contexto $G = \langle V_N, \Sigma, P, S\rangle$ tal que $L = \mathcal{L}(G)$. Definamos entonces:

$$G^* = \langle V_N \cup \{S\}, \Sigma, P \cup \{S' \to SS',\ S' \to \varepsilon\}, S'\rangle$$

Veamos que $\mathcal{L}(G^*) = L^*$. Esta demonstración es símilar a la anterior. La única defirencia es que en algún momento, en vez de usarse la regla $S' \to S$, se usa la regla $S' \to \varepsilon$.

- Si L_1 y L_2 son lenguajes independientes del contexto, entonces $L_1 \cap L_2$ no siempre es lenguaje independiente del contexto.

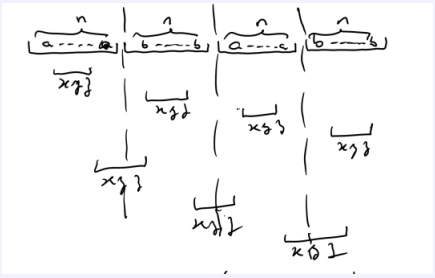
DEMOSTRACIÓN

Sean $L_1 = \{a^n b^m c^l : n, m, l \geq 0 \wedge n = m\}$ y $L_2 = \{a^n b^m c^l : n \geq 0 \wedge m = l\}$ dos lenguajes independientes de contexto (Demostrar?). Pero $L_1 \cap L_2 = \{a^n b^m c^l : n, m, l \geq 0 \wedge n = m = l\}$ no es lenguaje independiente como puede demostrarse utilizando el lema de pumping.

- El lenguaje $L = \{ww : w \in \Sigma^*\}$ no es independiente del contexto.

DEMOSTRACIÓN

Sea $L_1 = \{a^n b^m a^n b^m : n, m \geq 0\}$ es tal que $L_1 = L \cap a^* b^* a^* b^*$ donde $a^* b^* a^* b^*$ es un lenguaje regular.
La interesección entre un lenguaje regular y un lenguaje independiente de contexto da como resultado un lenguaje independiente de contexto (Demostrar?).
Entonces, si L fuera independiente del contexto, L_1 tambien debiera serlo.
Veamos que L_1 no es independiente de contexto:
Analizemos las componsiciones disponibleas para la cadena $a^n b^n a^n b^n \in L_1$:



En los primeros cuatro casos se desbalancean la cantidad de a o b, respectivamente y en el último caso se desbalancean la cantidad de a y b simultaneamente.

Lenguajes libres de contexto deterministicos

Un lenguaje L es libre de contexto deterministicos si existe un autómata de pila determinista A tal que $L = \mathcal{L}(A)$.

Para ver más sobre lenguajes libres de contexto deterministicos, ver Wikipedia.

Lema 1.7.2. El conjunto de lenguajes libres de contexto deterministicos es cerrado por la complementación.

Teorema 1.7.1. Para todo automáta de pila M, existe otro automáta de pila M' equivalente que siempre consume toda la cadena de entrada.

Teorema 1.7.2. El complemento de un lenguaje independiente del contexto deterministico es un lenguaje independiente del contexto deterministico.

Teorema 1.7.3. Existe un lenguaje independiente del contexto que es **no-deterministíco**

DEMOSTRACIÓN

Consideremos el lenguaje $L = a^k b^k c^k, k \geq 0$. Sabemos que L no es independiente del contexto. Su complemento puede escribirse como:

$$\bar{L} = \{a^i b^j c^k : i, j, k \geq 0 \wedge i \neq j\} \cup \{a^i b^j c^k : i, j, k \geq 0 \wedge j \neq k\} \cup \overline{a^* b^* c^*}$$

Osea que $\overline{L}$ puede escribirse como la unión de dos lenguajes independientes del contexto y un lenguaje regular. Es, por lo tanto, independiente del contexto. Supongamosq que $\overline{L}$ es un lenguaje independiente del contexto deterministico. Entonces, $\overline{\overline{L}}$ también debería serlo por el teorema anterior. Pero $\overline{\overline{L}} = L$ que ni siquiera es un lenguaje independiente del contexto. Luego $\overline{L}$ no es deterministico.

1.7.5. Gramáticas propias

Conjunto de símbolos activos

Símbolo alcanzable: Dada una grámatica $G = \langle V_N, V_T, P, S\rangle$, un símbolo $A \in V_N$ es alcanzable si existe una forma sentencial que lo contiene, $S \overset{*}{\Rightarrow} \ldots A \ldots$.

Símbolo activo: Dada una grámatica $G = \langle V_N, V_T, P, S\rangle$ un símbolo $A \in V_N$ es activo si existe $\alpha \in V_T^*$ tal que $A \overset{*}{\Rightarrow} \alpha$.

Símbolo útil: Un no terminal A es útil si y solo si es parte de una forma sentencial que genera una cadena de terminales, osea, si $S \overset{*}{\Rightarrow} \alpha A \beta \overset{*}{\Rightarrow} \omega$ con $\omega \in \Sigma^*$ y $\alpha, \beta \in (V_T \cup V_N)^*$.

Algoritmos para conseguir el conjunto de no-terminales activos de una gramática:

function ACTIVOS($\langle V_N, V_T, P, S\rangle$: Gramática)
  `Act` $\leftarrow \emptyset$
  repeat
    for $A \to \alpha \in P$ **do**
      if $\alpha \in (\mathtt{A} \cup V_T)^*$ **then**
        `Act` $\leftarrow$ `Act` $\cup \{A\}$
      end if
    end for

until `Act` no cambie
return `Act`
end function

Conjunto de símbolos anulables

Lema 1.7.3. Sea G una gramática libre de contexto. Entonces existe una grámatica propia (sin producciones borradoras) G' tal que genere el mismo lenguaje que G sin la cadena nula.

No terminal anulable: Un no terminal A es anulable si y solo $A \stackrel{*}{\Rightarrow} \varepsilon$.

Algoritmo para conseguir el conjunto de no terminales anulables de una gramática:

function ANULABLES($\langle V_N, V_T, P, S\rangle$: Gramática)
  $\texttt{An} \leftarrow \emptyset$
  for $A \to \alpha \in P$ **do**
    if $\alpha = \lambda$ **then**
      $\texttt{An} \leftarrow \texttt{An} \cup \{A\}$
    end if
  end for
  repeat
    for $A \to \alpha \in P$ **do**
      if $\alpha \in \texttt{An}^*$ **then**
        $\texttt{An} \leftarrow \texttt{An} \cup \{A\}$
      end if
    end for
  until `An` no cambie
  return `Anul`
end function

Gramáticas reducidas

Grámatica reducida: Una gramática $G = \langle V_N, V_T, P, S\rangle$ es reducida si y solo si para todo $A \in V_N$ se cumple que A es alcanzable y activo.

Algoritmo para conseguir una gramática propia:

function PROPIA($G = \langle V_N, V_T, P, S\rangle$: Gramática)
  $\texttt{An} \leftarrow \texttt{anulables}(G)$
  for $A \to \alpha \in P$ **do**
    if $\alpha = X_1 \dots X_k$ con $X_{j_1}, \dots, X_{j_m} \in \texttt{An}$ **then**
      Agregar todas las producciones $A \to \alpha'$ donde las α' se obtiene eliminando cada subconjunto en $X_{j_1}, \dots, X_{j_m}$ de α. Si todos los símbolos de α son anulables, no agregar $A \to \lambda$.
    end if

```
        if α = λ then
            Eliminar la producción A → λ
        end if
    end for
    return G modificada
end function
```

Forma normal de Chomsky

Una gramática $G = \langle V_N, V_T, P, S\rangle$ está en forma normal de Chomsky si y solo si todas sus producciones son de la forma $A \to BC$ o $A \to a$, donde $A, B, C \in V_N$ y $a \in V_T$.

Si L un lenguaje independiente del contexto tal que $\lambda \notin L$ entonces existe una gramática G en forma normal de Chomsky.

Forma normal de Greibach

Una gramática $G = \langle V_N, V_T, P, S\rangle$ está en forma normal de Greibach si y solo si todas sus producciones son de la forma $A \to a\alpha$ donde $A \in V_N$, $a \in V_T$ y $\alpha \in V_N^*$.

Si L un lenguaje independiente del contexto tal que $\lambda \notin L$ entonces existe una gramática G en forma normal de Greibach.

1.7.6. Otras propiedades

- Si $M = \langle Q, \Sigma, \delta, q_0, Z_0, F\rangle$ es un autómata de pila y L el lenguaje tal que $L = \mathcal{L}_\lambda(M)$, entonces L es independiente del contexto.
- Sea $G = \langle V_N, V_T, P, S\rangle$ una grámatica independiente del contexto. Se puede construir un autómata de pila M tal que $\mathcal{L}(G) = \mathcal{L}_\lambda(M)$.

1.8. Máquinas de Turing

El modelo más básico de una máquina de turing consiste en un control finito, una cinta infinita dividida en celdas y una cabeza de lectura. Cada celda de la cinta puede contener un único símbolo del alfabeto finito de la cinta.

Inicialmente, la cinta contiene una cadena de símbolos de entrada, seguida de un símbolo especial llamado blanco. La cabeza de lectura se coloca sobre el primer símbolo de la cadena de entrada. La máquina de Turing puede leer y escribir símbolos en la cinta o mover la cabeza de lectura a la izquierda o a la derecha.

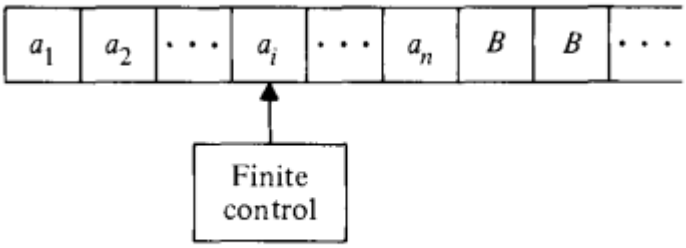


Fig. 7.1 Basic Turing machine.

Formalmente, definimos la máquina de Turing (MT) M como una tupla:

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$$

Donde:

- Q es un conjunto finito de estados.
- Σ es un alfabeto de entrada.
- Γ es un alfabeto de cinta.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ es una función de transición. Podría estar indefinido para algunos argumentos. La función indica devuele el estado al que se debe pasar, el símbolo a escribir en la posición actual de la cinta y el movimiento del cabezal (izquierda o derecha).
- $q_0 \in Q$ es el estado inicial.
- $B \in \Gamma$ es el símbolo blanco.
- $F \subseteq Q$ es el conjunto de estados finales.

Configuración instantanea: Una configuración instantánea de una MT es una tupla $\alpha_1 q \alpha_2$ donde $q \in Q$ y $\alpha_1, \alpha_2 \in \Gamma^*$ donde:

- α_1 son los simbolos de la cinta a la izquierda del cabezal.

- q es el estado actual.
- α_2 son los símbolos de la cinta a la derecha del cabezal hasta el último símbolo distinto de B.

Notemos que $\alpha_1, \alpha_2 \in \Gamma^*$, osea que pueden tener apariciones de B.

Asumismos que el cabezal está escaneando el símbolo más a la derecha de α_2 o un blanco si $\alpha_2 = \lambda$.

Movimiento del cabezal: El movimiento del cabezal se define como sigue: Sea

$$X_1 \dots X_{i-2} q X_i \dots X_n$$

una configuración instantánea de una MT M.

Supongamos que $\delta(q, X_i) = (p, Y, L)$:

- Si $i - 1 = n$, asumimos $X_i = B$
- Si $i = 1$ entonces es imposible moverse a la izquierda.
- Si $1 < i < n$ entonces escribimos:

$$X_1 \dots X_{i-1} q X_i \dots X_n \underset{M}{\vdash} X_1 \dots X_{i-2} p X_{i-1} Y X_{i+1} \dots X_n$$

  En caso de haya algún sufijo de $X_{i-1} Y X_{i+1} \dots X_n$ que sea completamente blanco, entonces se elimina.

Similarmente, si $\delta(q, X_i) = (p, Y, R)$:

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

Si dos configuracióones instantáneas están relacionadas por $\underset{M}{\vdash}$, entonces decimos que la segunda es un resultado de la primera.

Si una configuración instantánea resulta de otra en un número fínito de pasos, entonces están relacionadas por el símbolo $\overset{*}{\underset{M}{\vdash}}$.

Lenguaje aceptado por una MT: Definimos al lenguaje aceptado por una MT $M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$ como:

$$\mathcal{L}(M) = \{\omega \in \Sigma^* : q_0 \omega \overset{*}{\underset{M}{\vdash}} \alpha_1 p \alpha_2 \text{ con } p \in F \wedge \alpha_1, \alpha_2 \in \Gamma^*\}$$

Asumimos que la máquina de Turing se detiene cuando el input es aceptado. Además es posible que nunca se detenga si el input no es aceptado.

1.8.1. Autómatas linealmente acotados

Es una máquina de Turing no deterministica que cumple con las siguientes condiciones:

1. El alfabeto de entrada incluye dos símbolos especiales (¢ y \$) que son utilizados como topes izquierdo y derecho, respectivamente.

2. El autómata no se mueve ni a la izquierda de ¢ni a la derecha de \$, ni los sobreescribe.

Máquinas de dos cintas

Una máquina de Turing con dos o más cintas puede simularse usando la defición dada anteriormente. Para ello, definimos los estados en Γ como tuplas $[x_1, x_2]$ donde x_i sería el símbolo guardado en la i-ésima cinta.

Teorema 1.8.1. Si L es un lenguaje dependiente del contexto, entonces existe un autómata linealmente acotado A tal que $\mathcal{L}(A) = L$.

DEMOSTRACIÓN

Se construye el autómata linealmente acotado M tal que:

1. La primera cinta contiene la cadena de entrada $¢\omega\$$.

2. La segunda cinta se utiliza para generar las formas de la derivación. En cualquie instante, esta cinta contendrá la forma sentencial α que representa la derivación de la cadena de entrada. Se inicializa con el símbolo distingido S.

El autómata funcionará de la siguientes manera:

1. Si $\omega = \lambda$, entonces M se detiene rechanzando la cadena de entrada.

2. Sino:

   a) Selecciona (en forma no deterministica) la posición i dentro de α (la derivación que se encuentra en la cinta)

   b) Selecciona (en forma no deterministica) la producción $\beta \to \gamma \in P$.

   c) Si β aparece a partir de la posición i en α, entonces remplaza β por γ en α.

   d) Si la nueva forma sentencial α es tal que $|\alpha| > |\omega|$, entonces M se detiene rechazando la cadena de entrada.

   e) Sino, comparamos α con ω. Si son iguales, entonces M se detiene aceptando la cadena de entrada. Sino, se vuelve a repetir el paso 2.

1.8.2. Lenguajes recursivos

Son lenguajes L sobre un alfabeto Σ para el cual existe una máquina de Turing que se detiene para todo $\alpha \in \Sigma^*$ y la acepta (termina en un estado final) si $\alpha \in L$ o la rechaza (termina en un estado no final) si $\alpha \notin L$. En otras palabras, son los lenguajes para los cuales existe un algoritmo que permite decidir la pertenencia (o no) de toda cadena al lenguaje.

Teorema 1.8.2. Todo lenguaje dependiente del contexto es recursivo

DEMOSTRACIÓN

Sea $G = \langle N, \Sigma, P, S\rangle$ un gramática dependiente del contexto y $\omega \in \Sigma^*$. Vamos a construir un grafo fínito en el cual cada nodo tiene asocida un $\alpha \in (V_N \cup V_T)^+$ con $|\alpha| \leq |\omega|$. Las arista representas una producción de G, osea si $\gamma_1\tau\gamma_2$ y $\gamma_1\beta\gamma_2$ son dos nodos, entonces van a estar conectados si y solo si $\tau \to \beta \in P$. Por ser G dependiente del contexto, vale $|\tau| \leq |\beta|$, entonces $|\gamma_1\tau\gamma_2| \leq |\gamma_1\beta\gamma_2|$. Por lo tanto, el grafo es acotado.
En este grafo finito, es cierto que $S \overset{*}{\underset{G}{\Rightarrow}} \omega$ si y solo si existe un camino desde el nodo correspondiente a S hasta el nodo correspondiente a ω. Y esto último es decidible, o sea, tiene algoritmo. Por lo tanto, G es recursivo.

Lema 1.8.1. Sea $M_1, M_2, \ldots$ una enumeración de un conjunto de máquinas de Turing que paran para todas las entadradas. Siempre existe un lenguaje recursivo que no es aceptado por ninguna de ellas, o sea, siempre existe un lenguaje L tal que L es recursivo y $L \notin \mathcal{L}(M_i)$ para todo i.

DEMOSTRACIÓN

Consideremos el lenguaje definido $L = \{\omega_i : \omega_i \notin \mathcal{L}(M_i)\}$, es decir L está formado por todas las cadenas que son rechazadas por alguna de las máquinas M_i. Como es decidible si $\omega_i \in L$ o no (pues simplemente corremos las M_i con entrada ω y vemos si para en un estado no final), entonces L es recursivo.
Supongamos ahora que este lenguaje es aceptado por alguna de las máquinas enumeradas. Sea M_j, la misma. Entonces ¿que pasa si $\omega_j \in L$?
Por definición de L tenemos que $w_j \in L \iff w_j \notin \mathcal{L}(M_j) \iff w_j \notin L$. Esto es un absurdo, por lo tanto L no es aceptado por ninguna máquina $M_1, M_2, \ldots$.

Lema 1.8.2. Existe un lenguaje recursivo que no es dependiente del contexto

DEMOSTRACIÓN

Vamos a tratar de demostrar que podemos encontrar una numeración de máquinas Turing correspondientes a cada uno de los lenguajes dependientes del contexto definidos sobre $\{0, 1\}^*$.

Estas máquinas de Turing paran en todas las entradas porque como los lenguajes son dependientes de contexto, siempre hay un algoritmo de reconomcimiento cuando el lenguaje es dependiente del contexto.
Codifiquemos todas estas gramaticas con cadenas binarias, es decir, demos una representación binaria a cada símbolo de la gramática codificando:

- $0 = 10$
- $1 = 100$
- y al resto de los simbolos $s_k = 10^k$

Gracias a esta codificación mediante 0s y 1s, podemos enumerarlas de la siguientes manera: $G_1, G_2, \ldots$. Además, por el teorema 1.8.2, existen autómatas $M_1, M_2, \ldots$ que aceptan los lenguajes generados por cada una de ellas.
Luego, por el lema, anterior, tenemos que existe un lenguaje L recursivo que no es aceptado por ninguno de estos autómatas y no es dependiente del contexto.

1.8.3. Máquinas de Turing No Deterministicas

Teorema 1.8.3. Sea M una máquina de Turing no deterministica y $L = \mathcal{L}(M)$. Entonces existe una máquina de Turing determinista M' que acepta el mismo lenguaje.

DEMOSTRACIÓN

Sea r la máxima cantidad posible de transiciones que parte de un estado cualquiera de M. Numeremos, para cada estado, las transiciones que parten de él con números entre 1 y r.
Entonces toda secuencia finita de enteros entre 1 y r representa un camino en M partiendo desde el estado inicial q_0. Algunas de estas secuencias no serán ejecutables por no existir alguna de sus transiciones.
Podemos construir una máquina de turing M' de tres cintas tal que:

- La primera contiene la entrada de M
- La segunda, una secuencia de enteros entre 1 y r que corresponderá a una secuencia de transiciones a partir del estado inicial q_0.
- La tercera servirá para simular la M.

Y funcionará de la siguiente manera:

- Generamos secuencias de enteros con valroes entre 1 y r en orden creciente de longitud y alfabetico, que irán siendo escritos en la segunda cinta.
- Para cada secuencia:

1. Borramos el contenido de la cinta 3
2. Copiamos el contenido de la cinta 1 en la cinta 3.
3. Simulamos la MT en la cinta 3 ejecutando la secuencia de transiciones en la cinta 2.
4. Aceptamos la cadena de la cinta 1 si se acepta esa cadena en la simulación que corremos en la cinta 3.

- Cuando la cadena analizada no es aceptada por M, M' no parará.

Teorema 1.8.4. Sea L el lenguaje de la gramática $G = \langle V_N, V_T, P, S \rangle$. Entonces existe una máquina de Turing M que acepta el mismo lenguaje.

DEMOSTRACIÓN

Podemos construir una máquina de Turing M no deterministica de dos cintas tal que:

- la primera contiene la cadena de entrada ω
- La segunda contiene la forma setencial α de la derivación de la cadena de entrada. Se inicializa con el símbolo inicial de la gramática S.

Y opera de la siguiente manera:

1. Seleccionar (en forma no-deterministica) la posición i dentro de α
2. Seleccionar de forma no-deterministica una regla de producción $\beta \to \gamma \in P$
3. Si β aparece a partir de la posición i en $alpha$, entonces reemplazar β por γ en $alpha$.
4. Comparar la nueva forma sentencial α con la cadena de entrada ω. Si son iguales, entonces aceptar la cadena de entrada. Si no, volver al paso 1.

Teorema 1.8.5. Sea L el lenguaje aceptado por una máquina de Turing M. Entonces existe una gramática sin restricciones G que genera el mismo lenguaje.

DEMOSTRACIÓN

Vamos a armar una gramática que genere dos copias de alguna representación de la cadena de entrada y que luego simule la operación de la maquina de Turing sobre una de ellas. Si esta simulación resulta en una aceptación, entonces la primera copia se convierte en la cadena igual a la de entrada.

Entonces sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F\rangle$ una máquina de Turing que acepta el lenguaje L. Definimos la gramática $G = \langle V_N, \Sigma, P, A_1\rangle$ de la siguiente manera:

- $V_N = ((\Sigma \cup \{\lambda\}) \times \Gamma) \cup \{A_1, A_2, A_3\}$
- P contiene las siguientes producciones:
  - $A_1 \to q_0 A_2$
  - $A_2 \to [a, a]A_2$ para cada $a \in \Sigma$
  - $A_2 \to A_3$
  - $A_3 \to [\lambda, B]A_3$
  - $A_3 \to \lambda$
  - $q[a, X] \to [a, Y]p$ para cada $q \in Q, a \in \Sigma \cup \{\lambda\}, X, Y \in \Gamma$ tales que $\delta(q, X) = (p, Y, R)$
  - $[b, Z]q[a, X] \to p[b, Z][a, Y]$ para todo $q \in Q, a, b \in \Sigma \cup \{\lambda\}, X, Y, Z \in \Gamma$ tales que $\delta(q, X) = (p, Y, L)$
  - $[a, X]q \to qaq$, $q[a, X] \to qaq$, $q \to \lambda$ para todo $q \in F, a \in \Sigma \cup \{\lambda\}, X \in \Gamma$

Utilizando las reglas 1 y 2 se puede generar $A_1 \overset{*}{\Rightarrow} q_0[a_1, a_1] \dots [a_n, a_n]A_2$
Luego, con la regla 3 se generan los símbolos correspondientes a los a los espacios en blancos necesarios para el analísis de la cadena de entrada en M:

$$A_1 \overset{*}{\Rightarrow} q_0[a_1, a_1] \dots [a_n, a_n][\lambda, B]^m A_3$$

Utilizando las reglas 6 y 7 se simula la operación de la máquina de Turing sobre las segundas componentes, dejando intactas las primeras componentes.
Veamos que:

$$q_0 a_1 \dots a_1 \overset{*}{\underset{M}{\vdash}} X_1 \dots X_{r-1} q X_r \dots X_s \Rightarrow$$

$$q_0[a_1, a_1] \dots [a_n, a_n][\lambda, B] \overset{*}{\underset{G}{\Rightarrow}} [a_1, X_1] \dots [a_{r-1,X_{r-1}}] q[a_r, X_r] \dots [a_{n+m}, X_{n+m}]$$

con $a_1, \dots, a_n \in \Sigma$, $X_1, \dots, X_{n+m} \in \Gamma$, $a_{n+1} = \dots = a_{n+m} = \lambda$, $X_{n+1} = \dots = X_{n+m} = B$.
Para $\overset{0}{\vdash}$ vale trivialmente pues:

- $q_0 a_1 \dots a_n \overset{0}{\vdash} q_0 a_1 \dots a_n$
- y $q_0[a_1, a_1] \dots [a_n, a_n][\lambda, B]^m \overset{0}{\Rightarrow} q_0[a_1, a_1] \dots [a_n, a_n][\lambda, B]^m$

Ahora, probemos por introduccion que vale lo mismo para $\overset{k}{\vdash}$ si vale para $\overset{k-1}{\vdash}$. Supongamos que $q_0 a_1 \dots a_n \overset{k}{\vdash} q_0 a_1 \dots a_n$. Entonces:

Suponiendo que vale

$$q_0 a_1 \dots a_1 \overset{k-1}{\underset{M}{\vdash}} X_1 \dots X_{r-1} q X_r \dots X_s \Rightarrow$$

$$q_0[a_1, a_1] \dots [a_n, a_n][\lambda, B] \overset{*}{\underset{G}{\Rightarrow}} [a_1, X_1] \dots [a_{r-1,X_{r-1}}] q [a_r, X_r] \dots [a_{n+m}, X_{n+m}]$$

Si en el paso k, el movimiento es hacia la derecha, o sea $\delta(q, X_r) = (p, Y_r, R)$, por la regla 6 sabemos que $q[a_r, X_r] \to [a_r, Y_r]p \in P$, por lo que

$$[a_1, X_1] \dots [a_{r-1,X_{r-1}}] q [a_r, X_r] \dots [a_{n+m}, X_{n+m}] \overset{*}{\Rightarrow}$$
$$[a_1, X_1] \dots [a_r, Y_r] p [a_{r+1}, X_{r+1}] \dots [a_{n+m}, X_{n+m}]$$

Si en el paso k, el movimiento es hacia la izquierda, o sea $\delta(q, X_r) = (p, Y_r, L)$ entonces por la regla 7 sabemos que $[a_{r-1}, X_{r-1}]q[a_r, X_r] \to p[a_{r-1}, X_{r-1}][a_r, Y_r] \in P$ por lo que:

$$[a_1, X_1] \dots [a_{r-1,X_{r-1}}] q [a_r, X_r] \dots [a_{n+m}, X_{n+m}] \overset{*}{\Rightarrow}$$
$$[a_1, X_1] \dots p [a_{r-1}, Y_{r-1}][a_r, X_r] \dots [a_{n+m}, X_{n+m}]$$

Entonces, si llegamos a una forma sentencial en la que el estado q sea final, aplicando repetidas veces la regla 8 tenemos que queda cualquiera de las dos derivaciones va a derivar en algo del estilo $a_1 a_2 \dots a_n$.
En el sentido inverso, tambien puede demostrarse que si $\alpha \in \mathcal{L}(G)$ entonces $x \in \mathcal{L}(M)$

1.8.4. Lenguajes recursivamente enumerables

Los lenguajes recursivamente enumerables son lenguajes para los que existe una máquina de turing que puede enumerar todas las cadenas válidas del mismo. Si el lenguaje es infinito, entonces se puede elegir un algoritmo que evite repeticiones.

Los lenguajes regulares, los libres de contexto, los dependientes del contexto y los recursivos son lenguajes recursivamente enumerables.

Codificación de máquinas de Turing restringidas a u alfabeto binario

Sea $M = \langle Q, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_2\}\rangle$ una máquina de Turing con $Q = \{q_1, q_2, \dots, q_n\}$ el conjunto de estados. Entonce, si usamos $X_1 = 0$, $X_2 = 1$, $X_3 = B$, $D_1 = L$ y $D_2 = R$. Entonces podemos codificar cada transición $\delta(q_i, X_j) = (q_k, X_l, D_m)$ como una cadena de la forma $0^i 1 0^j 1 0^k 1 0^l 1 0^m 1$. Y el código binario para la máquina de Turing M es la concatenación de todas las cadenas de transiciones de la siguiente forma: $111 trans_1 11 \dots 11 trans_r 111$.

Teorema 1.8.6. Existe un lenguaje que no es recursivamente enumerable.

DEMOSTRACIÓN

Ordenemos el conjunto $\{0,1\}^*$ primero por longitud y luego siguiendo el orden léxico. Llamemos w_i a la $i-$ésima cadnea según este ordenamientos y M_j a la máquina de Turing cuyo código binario es el entero positivo j.
Entonces podemos construir una tabla que indique con un 1 en la entrada i,j si $w_i \in \mathcal{M}_|$ y un 0 cuando no.
Definamos el lenguaje L_d como

$$L_d = \{w_i \mid w_i \notin \mathcal{L}(M_i)\}$$

que corresponde a los ceros de la diagonal de la tabla.
Si L_d es recursivamente enumerable, entonces existe una máquina de Turing M_j tal que $\mathcal{L}(M_j) = L_d$. Pero entonces, $w_j \notin L_d \iff w_i \notin \mathcal{L}(M_i) \iff w_j \notin L_d$ lo que es una contradicción.
Luego L_d no es recursivamente enumerable.

Lenguaje universal: Definimos L_u como el lenguaje constituido por todos los pares de máquinas de Turing M- cadena de entrada ω tales que $\omega \in \mathcal{L}(M)$:

$$L_u = \{(M,\omega) \mid \omega \in \mathcal{L}(M)\}$$

Teorema 1.8.7. L_u es recursivamente enumerable.

DEMOSTRACIÓN

Para demostrarlo vamos a crear una máquina de Turing de 3 cintas que acepta L_u de la siguiente manera:

- La cinta 1 contiene la cadena de entrada, o sea, la codificación de la máquina M seguida de la cadenaω.
- La segunda cinta se utiliza para simular la cinta de la máquina de turing original.
- La tercera cinta contiene el estado actual de la máquina de turing simulada.

La máquina funciona de la siguiente mánera:

1. Chequear que la cadena que codifica la máquina turing de M en la cinta 1 sea sintácticamente correcta y determinística.
2. Inicilizar la cinta 2 con la cadena de entrada y la cinta 3 con 0 (que representa el estado inicial q_1). Todas las cabezas deben posicionarse en la posición extrema izquierda de sus respectivas cintas.

3. Simular la máquina de turing

   a) Si el contenido de la cinta 3 es 00 (correspondiente al estado q_2), entonces terminar y aceptar.

   b) Sea X_j es el símbolo actualmente leído en la cinta 2 y sea 0^i el contenido de la cinta 3 (estado actual q_i), entonces:

      1) Comenzando desde la posición extrema izquierda, buscar en la cinta 1 la cadena 110^i10^j1 (correspondiente a $d(q_i, X_j)$) hasta encontrarla o hasta encontrar la cadena 111.
      2) Si se encontró la cadena 111 entonces parar y rechazar
      3) Si se encontró la cadena 110^i10^j1 entonces esto indicará que la transición que se debe realizar es $d(q_i, X_j) = (q_k, X_l, D_m)$, osea $0^i10^j10^k10^l10^m$. Entonces poner en la cinta 3 0^k y en la cinta 2 X_l en la posición actual y moverse en la dirección D_m.
      4) Volver al paso 3.*a*

Teorema 1.8.8. El lenguaje L_u no es recursivo

DEMOSTRACIÓN

Supongamos que existe un algoritmo para reconocer L_u entonces podriamos construir un algoritmo para reconocer su complemento que es el leguaje L_d. Pero entonces L_d sería recursivamente enumerable. Absurdo

Capítulo 2

La que da Vero

2.1. Transformación de gramáticas

Dada una expresión en un lenguaje formal, buscamos encontrar un algoritmo que permita, a la computadora, entenderla. Para esto debemos realizar los siguientes pasos:

1. Revisar que la expresión solamente tenga símbolos del alfabeto del lenguaje.
2. Revisar que la expresión tenga una forma que respete la grámatica.
3. Mapear la expresión, usando la grámatica, a una estructura de datos que pueda ser procesada por la computadora.

Los lenguajes que no son recursivamente enumerables no pueden ser reconocidos por ninguna máquina de Turing.

Los lenguajes regulares y algunos lenguajes libres de contexto se pueden analizar en tiempo lineal con respecto al tamaño de la entrada.

El analísis de los demás lenguajes libres de contexto se puede realizar en tiempo polinomial con respecto al tamaño de la entrada. Y para lenguajes arbirtrarios no hay cota en la comlejidad computaciónal para analizarlos.

2.1.1. Definiciones para recordar

Alfabeto: Conjunto finito de símbolos.

Cadena: Secuencia finita de símbolos de un alfabeto.

Lenguaje: Conjunto de cadenas.

Hay lenguajes naturales y lenguajes formales. Los segundos se definen matemáticamente mediante grámaticas y/o autómatas.

Grámatica: Una grámatica formal es una cuádrupla $G = (N, T, S, P)$ donde:

N es un conjunto finito de símbolos no terminales (Variables).

T es un conjunto finito de símbolos terminales (Constantes).

S es un símbolo distinguido de Nno terminal que representa el símbolo inicial.

P es un conjunto finito de reglas de producción de la forma $A \to \alpha$, donde $A, \alpha \in (N \cup T)^*$.

Derivación: Sea $G = (N, T, S, P)$ una grámatica formal. Si $\alpha, \beta, \gamma_1, \gamma_2 \in (N \cup T)^*$ y $\alpha \in \beta$, entonces

$$\gamma_1 \alpha \gamma_2 \to \beta$$

.

La relación $\Rightarrow$ es el subconjunto $N \cup T)^* \times (N \cup T)^*$ y significa derivar en un solo paso.
Las relacioens $\overset{+}{\Rightarrow}$ y $\overset{*}{\Rightarrow}$ son la clausura transitiva y reflexo-transitiva de $\Rightarrow$, respectivamente.
El lenguaje generado por la grámatica G es:

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

Jerarquía de Chomsky

Es una clasifición jerarquica de tipos de grámaticas formales que generan lenguajes formales:

- **Tipo 0**: Gramáticas sin restricciones. Son todos los lenguajes aceptados por una máquina de Turing y se los llama lenguajes recursivamente enumerables.
- **Tipo 1**: Gramáticas sensibles al contexto. Las reglas son de la forma $\alpha \to \beta$ con $|\alpha| < |\beta|$ y $\alpha, \beta \in (N \cup T)^*$.

  Son aceptados por autómatas linealmente acotados que son máquinas de Turing determínisticas cuya cinta de memoria está acotada por un múltiiplo de la longitud de entrada.

  Reconocer este tipo de lenguajes es un problema PSPACE-completo.
- **Tipo 2:** Gramáticas libres de contexto. Las reglas son de la forma $A \to \gamma$ con A un no terminal y γ una cadena de no terminales y terminales.

  Estas grámaticas generan todos los lenguajes aceptados por automátas de pila.

  No es cierto que para cada autómata de pila no deterministico exista uno deterministico equivalente.

  Los lenguajes libres de contexto aceptados por autómatas de pila deterministicos se llaman lenguajes LR y se los puede reconocer en tiempo lineal en el tamaño de la entrada.

  Un subconjunto propio de los lenguajes LR son los generados por grámaticas LL y el algoritmo para reconocerlos es muy sencillo.

  Para reconocer un lenguaje libre de contexto no determínistico hay un algoritmo de complejidad cúbica en el tamaño de la entrada.

- **Tipo 3:** Gramáticas regulares. Las reglas son de la forma $A \to a$ ó $A \to aB$ con A, B símbolos no terminales y a un símbolo terminal.

  Alternativamente, las reglas son de la forma $A \to a$ y $A \to Ba$ con A, B símbolos no terminalesY a un terminal. La regla $S \to \lambda$ tambíen está permitida siempre y cuando S no aparezca en la parte derecha de ninguna regla.

  Estas gramáticas generan todos los lenguajes aceptados por autómatas finitos determinísticos o no determísticos.

  Para determinar si una cadena pertenece o nó a un lenguaje regular, siempre hay un algoritmo lineal en el tamaño de la cadenta.

Árbol de derivación

Un árbol de derivación es una representación gráfica de una derivación que no muestra el orden en que se aplicaron las producciones. LAs estiquetas de los nodos están en $N \cup T \cup \{\lambda\}$. La de los nodos internos son no terminales y la de sus hijos son los símbolos del cuerpo de una producción.

Si un nodo tiene una etiqueta A entonces hay una producción $A \to X_1 \dots X_n$ y el nodo tiene n descendientes etiquetados $X_1, \dots, X_n$.

Cada árbol de derivación tiene asociada una única derivación más a la izquierda y una única derivación más a la derecha.

2.1.2. Análisis síntactico o parsing

Sea $G = (N, T, S, P)$ una grámatica libre de contexto. Sea $\alpha \in (T \cup N)^*$. Un parsing más a la izquierda de α es la secuencia de producciones usadas en la derivación más a la izquierda de α desde S. Un parsin más a la derecha de α es la secuencia de producciones usadas en la derivación más a la derecha de α desde S.

Gramática ambigua: Una gramática G es ambigua si hay una expresión de $\mathcal{L}(G)$ que tiene más de una derivación más a la izquierda o más de una derivación más a la derecha.

Gramática no ambigua: Una grámatica libre de contexto es no ambigua si cada cadena de su lenguaje tiene una única derivación más a la izquierda.

Las gramáticas libres de contexto determinísticas son no ambiguas.

Decibilidad de un problema: Un problem que requiere de respuesta por si o por no, no es decidible cuando no hay ningún algoritmo capaz de responder todas las (infinitas) instancias de un problema.

El problema de decidir si una gramática libre de contexto es ambigua no es decidible.

Cualquier lenguaje no vacío admite una gramática ambigua: Para generarla se puede tomar una gramática no ambigua y agregarle una regla duplicada.

Hay lenguajes **inherementes ambiguos** que solo pueden ser generados por grámaticas ambiguas.

Si nos enfrentamos con una grámatica ambigua, podemos hacer dos cosas:

1. Cambiarla
2. Descartar árboles de derivación, dando reglas de precedencia.

Grámaticas recursivas a izquierda: Una grámatica $G = (N, T, S, P)$ libre de contexto, es recursiva a izquierda si tiene un no terminal A tal que para alguna expresión $\alpha \in (N \cup T)^*$,

$$A \underset{L}{\overset{*}{\Rightarrow}} A\alpha$$

Se llama recursión inmediata cuando hay una producción $A \to A\alpha$

Teorema 2.1.1. Todo lenguaje libre de contexto tiene un grámatica que no es recursiva a izquierda

Para demostrar este teorema daremos un algoritmo que transforma una grámatica G que es recursiva a izquierda en otra grámatica G' que no lo es y que ambas generan el mismo lenguaje.

Grámatica sin ciclos: Una grámatica no tiene ciclos si para todo símbolo no terminal A no hay derivaciónes $A \underset{L}{\overset{+}{\Rightarrow}} A$

En las gramáticas libres de contexto, los ciclos se originan en la existencia de producciones $A \to B$ con B un único no terminal y en las producciones $A \to \lambda$.

Es posible transformar las grámatica en otra sin ciclos eliminando las producciones $A \to \lambda$ (solamente podemos dejar $S \to \lambda$ y eliminando las producciones con un solo no-terminal en el cuerpo.

2.1.3. Eliminación de símbolos inutiles

Eliminación de símbolos inactivos

Los **símbolos inactivos** son los símbolos no terminales que no pueden generar una cadena de terminales.

Input: $G = (V_N, V_T, P, S)$ una grámatica libre de contexto.

Output: G' sin símbolos inactivos.

Algoritmo:

1. $Act \leftarrow \emptyset$
2. Mientras sigan apareciendo elementos nuevos en Act

$$Act \leftarrow Act \cup \{X \in V_N \mid X \to \gamma \in P \text{ con } \gamma \in (V_T \cup Act)^*\}$$

3. $G' \leftarrow (V_N \cap Act, V_T, P, S)$

Eliminación de símbolos inaccesibles

Los **símbolos inaccesibles** son los símbolos de una grámatica que no son usados en ninguna derivación.

Input: $G = (V_N, V_T, S, P)$ una grámatica libre de contexto tal que $\mathcal{L}(G) \neq \emptyset$ y sin símbolos inactivos.

Output: Una grámatica $G' = (V'_N, V'_T, P', S)$ tal que

- $\mathcal{L}(G') = \mathcal{L}(G)$
- y $\forall X \in (V'_N \cup V'_T),\ \exists \alpha, \beta \in (V'_N \cup V'_T)^*$ tal que $S \overset{*}{\Rightarrow} aX\beta$.

Algoritmo

1. $Alc \leftarrow \{S\}$
2. Mientras sigan apareciendo elementos en Alc:

   $$Alc \leftarrow Alc \cup \{X \in (V_T \cup V_N) \mid \exists A \rightarrow \alpha X \beta \in P \wedge A \in Alc\}$$

3. $G' = (Alc \cap V_N, Alc \cup V_T, S, P', S)$ con P' el conjunto de producciones de G que solo usan los símbolos en Alc.

2.1.4. Eliminación de ciclos

Eliminación de producciones λ

Input: $G = (V_N, V_T, S, P)$ una grámatica libre de contexto tal que $\mathcal{L}(G) \neq \emptyset$, sin símbolos inútiles.

Output: Una grámatica G' sin producciones lambda distintas de $S \rightarrow \lambda$.

Algoritmo:

1. $Nul \leftarrow \emptyset$
2. Mientras sigan apareciendo elementos nuevos en Nul:

   $$Nul \leftarrow Nul \cup \{X \in V_N \mid X \rightarrow \alpha \in P \text{ con } \alpha \in Nul^*\}$$

3. Remplazar cada producción de la forma $X \rightarrow \alpha_0 Y_1 \alpha_1 \ldots \beta_k Y_k$ donde $Y_i \in Nul$ y $a_i \in ((V_T \cup V_N) - NUL)^*$. Por $X \rightarrow \alpha_0\beta_1\alpha_1 \ldots \alpha_k\beta_k$ donde cada β_i es o bien Y_i o bien λ.
4. Eliminar todas las producciónes de la forma $X \rightarrow \lambda$.
5. Si $S \in Nul$ agregar un nuevo terminal inicial S' y las producciones $S' \rightarrow S|\lambda$.

Eliminación de producciónes cuyo cuerpo es un único no terminal

Input: $G = (V_N, V_T, S, P)$ una grámatica libre de contexto tal que $\mathcal{L}(G) \neq \emptyset$ y sin símbolos inaccesibles y sin producciones lambda.

Output: Una grámatica G' sin producciones cuyo cuerpo es un único no terminal.

Algoritmo:

1. Para cada símbolo $A \in V_N$ armar $N_A = \{B | A \overset{*}{\Rightarrow} B\}$ de la siguiente manera:

   $N_A \leftarrow \{A\}$

   Mientras sigan apareciendo elementos nuevos en N_A:

   $N_A \leftarrow N_A \cup \{C \in V_N \mid B \to C \in P \wedge B \in N_A\}$

2. Construir el conjunto de producciones para la nueva gramática de la siguiente forma:

   a) Para cada $B \in N_A$: Si $B \to \alpha \in P$ y no α no es un único terminal, entonces agregata $A \to \alpha$ a P'.

2.1.5. Eliminación de recursión izquierda

Algoritmo de la eliminación de recursión inmediata a izquierda

Sea $G = (N, T, S, P)$ una grámatica libre de contexto, donde P es el conjunto de producciones:

$$A \to A\alpha_1 | \ldots | A\alpha_n | \beta_1 \ldots \beta_m$$

donde ninguno del β empieza con A.

Definimos $G' = (N \cup A', T, P', S)$ con A' un nuevo símbolo no terminal y P' identico a P excepto que se reemplazan todas las producciones de lado izquiero A por:

$$A \to \beta_1 | \ldots | \beta_m | \beta_1 A' | \ldots | \beta_m A'$$

$$A' \to \alpha_1 | \ldots | \alpha_n | \alpha 1 A' | \ldots | \alpha_n A'$$

P' tiene el doble de producciones que P. El algoritmo tiene complejidad $|P|$.

Eliminación de la recursión a izquierda

Input: $G = (V_N, V_T, S, P)$ una grámatica libre de contexto tal que $\mathcal{L}(G) \neq \emptyset$ y sin símbolos inútiles y sin ciclos.

Output: Una grámatica G' sin recursión izquierda que acepta el mismo lenguaje que G.

Numeramos todos los símbolos no terminales $A_1, \ldots, A_n$.

Invariante del ciclo: Todas las producciones con cabeza A_k para $k = 1, \dots i-1$ fueron posbilemente transformadas a $A_k \to \alpha$ done α empieza con un terminal o un no-terminal mayor que A_k.

Algoritmo:

Para $i = 1, \dots, n$:

Para $j = 1, \dots, i-1$:

Si $A_i \to A_j\alpha$ y $A_j \to \delta_1|\delta_2'| \dots |\delta_m$:

Remplazar $A_i \to A_j\alpha$ por $A_i \to \delta_1|\delta_2'| \dots |\delta_m\alpha$.

Eliminar la recursion inmediata de A_i.

Análisis de complejidad: Sea c la máxima cantidad de producciones con la misma cabeza. En el peor caso tenemos:

- Todas las producciones de G con cabeza A_1 tiene recursión inmediata que con la transformación pasan a ser $2c$ producciones con cabeza A_1.
- Todas las producciones de G con cabeza A_2 tienen un cuerpo que comienza con A_1. Entonces, tenemos 2^2c^2 producciones con cabeza A_2.
- $\vdots$
- Todas las producciones de G con cabeza A_n tienen un cuerpo que comienza con A_1. Etnoces pasan a ser $(2c)^{2^n}$ producciones con cabeza A_n.

Entonces, en el peor caso tenemos $2(2c)^{2^n}$ producciones con cabeza A_n.

2.1.6. Transformación a Forma Normal de Chomsky

Sea $G = (V_N, V_T, S, P)$ una grámatica libre de contexto tal que $\mathcal{L}(G) \neq \emptyset$. Gestá en forma normal de Chomsky si cada una de sus producciónes es de la forma $A \to BC$ y $A \to a$, donde $A, B, C \in V_N$ y $a \in V_T$. Y en caso de que $\lambda \in \mathcal{L}(G)$, $S \to \lambda \in P$ y λ no aparece en ningún otro lado.

Algoritmo de transformación a forma normal de Chomsky

Input: $G = (V_N, V_T, S, P)$ una grámatica libre de contexto sin recursión a izquierda.

Output: Una grámatica $G' = (V_N, V_T, S, P')$ en forma normal de Chomsky que acepta el mismo lenguaje que G.

Algoritmo:

1. $P' \leftarrow \{A \to a \in P \mid A \in V_N \wedge a \in V_T\}$
2. $P' \leftarrow P' \cup \{A \to BC \mid A, B, C \in V_T\}$
3. Si $S \to \lambda \in P$ entonces:

   $P' \leftarrow P' \cup \{S \to \lambda\}$
4. Para cada producción de la forma $A \to X_1 \dots X_k$ con $k \geq 2$ agregamos a P' las siguientes producciones:

   $A \to X_1' \langle X_2 \dots X_k \rangle$
   $\langle X_2 \dots X_k \rangle \to X_2' \langle X_3 \dots X_k \rangle$
   $\vdots$
   $\langle X_{k-1} k \rangle$

   Donde $X_i, \dots, X_k$ son nuevos símbolos no terminales.
5. Para cada producción de la forma $A \to X_1 X_2$ tal que X_1 y/o X_2 son terminales, agreamos a P' la producción $A \to X_1' X_2'$.
6. Por cada no terminal de la forma a' introducido en los últimos dos pasos agregamos una producción $a' \to a$ (con $a \in V_T$).

2.1.7. Transformación a forma normal de Greibach

Sea $G = (V_N, V_T, S, P)$ una grámatica libre de contexto tal que $\mathcal{L}(G) \neq \emptyset$. Gestá en forma normal de Greibach si cada una de sus producciónes es de la forma $A \to a\alpha$ con $\alpha \in V_N$ y no hay producciones λ.

Algoritmo de transformación a forma normal de Greibach

Input: $G = (V_N, V_T, S, P)$ una grámatica libre de contexto sin recursión a izquierda.

Output: Una grámatica $G' = (V_N, V_T, S, P')$ en forma normal de Greibach que acepta el mismo lenguaje que G.

Algoritmo:

1. Contruimos una relación de orden entre todos los símbolos en V_N.
2. Para i desde $n-1$ hasta 0:

   Remplazamos cada produccion de la forma $A_i \to A_j\alpha$ con $(j > 1)$ por $A_i \to \beta_1\alpha| \dots |\beta_m\alpha$ donde $A_j \to \beta_1| \dots |\beta_m$.

3. En este momentos todas las producciónes empiezan con un terminal. Ahora, para cada producción $A \to aX_1 \dots X_n$ remplazamos todos los X_j que son terminales por un nuevo no terminal X'_j.

4. Para cada uno de los símbolos no terminales X'_j introducidos en el paso anterior, agregamos las producciones $X'_j \to X_j$