Lingüística Computacional

Víctor Mijangos de la Cruz

II. Perspectivas formales

Introducción a los lenguajes formales

Alfabeto

Alfabeto

Un alfabeto es un conjunto $\Sigma = \{a_1, a_2, ..., a_n\}$, donde cada $a_i \in \Sigma, i = 1, ..., n$, es un elemento al que llamamos *símbolo*.

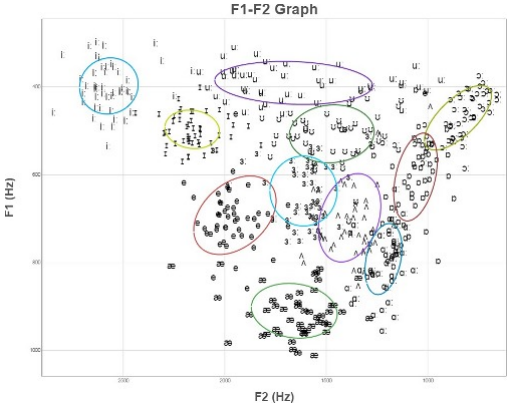
Por ejemplo:

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

es el alfabeto binario

Símbolos

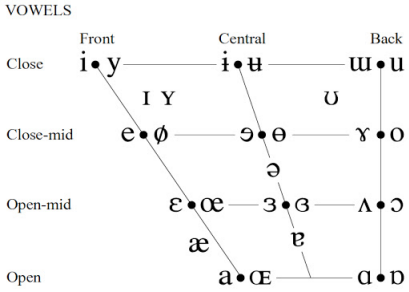
En el lenguaje natural, las palabras, oraciones, etc. está representados por sonidos.
Sin embargo, los sonidos del habla no son estáticos.



Símbolos

Los sonidos del lenguaje son abstraídos para representarse por medio de diferentes símbolos.

En lingüística, las abstracciones de los sonidos se llaman **fonemas**. Existen alfabetos, como el Alfabeto Fonético Internacional (AFI o IPA) que buscan representar los fonemas en todas las lenguas conocidas.



Símbolos

Cuando trabajamos con texto, sin embargo, los sonidos están representados por **letras** o caracteres. En general, utilizamos el alfabeto latino, pero debe tomarse en cuenta que existen otros tipos de alfabetos.

Podemos denotar este alfabeto como:

$$\Sigma = \{a, b, c, ..., z\}$$

El conjunto de símbolos que elijamos determinará las estructuras que podamos conformar en las cadenas. En el caso del lenguaje, los símbolos dependen de los **sistemas de escritura**.

Cadenas

Podemos definir el producto cartesiano entre alfabetos como:

- $\Sigma^1 = \Sigma$
- $\Sigma^{n+1} = \Sigma^n \times \Sigma$

Esto nos da arreglos ordenados de la forma $(a_1, a_2, ..., a_n)$. Entonces:

Cadena

Sea Σ un alfabeto. Definimos una *cadena sobre* Σ como una función $\overline{x} : n \to \Sigma^n, n \in \mathbb{N}$.

- La longitud de una cadena se denota como $|\overline{x}|$.
- El i-ésimo elemento de una cadena se denota $\overline{x}[i]$.

Operaciones sobre cadenas

Lo que buscamos con la teoría de lenguajes formales es describir todas las cadenas que puedan presentarse en un lenguaje.

Sin embargo, enumerar todas las cadenas es demasiado costoso. Por tanto, se buscan describir cómo se **generan** las cadenas (Clark, et al., 2013).

Por tanto, deben definirse operaciones que puedan generar cadenas nuevas a partir de cadenas existentes,

Concatenación

Sean $\overline{x} = (a_1, ..., a_n)$ y $\overline{y} = (b_1, ..., b_n)$ dos cadenas sobre el alfabeto Σ, entonces la concatenación de $\overline{x}$ con $\overline{y}$, denotada como $\overline{x} \cdot \overline{y}$, es la cadena $(a_1, ..., a_n, b_1, ...b_n)$.

Aquí ϵ es un símbolo vacío o neutral tal que $\epsilon \cdot w = w \cdot \epsilon = w$

Símbolo nulo

Definiremos un símbolo que funcionará como elemento nulo cuando operemos con cadenas, se conoce como ϵ (epsilon).

Elemento nulo

El elemento nulo ϵ cumple que, para toda cadena a, se tiene que:

$$\epsilon \cdot a = a \cdot \epsilon = a$$

También definiremos la siguiente operación sobre el alfabeto:

$$\Sigma^0 = \{\epsilon\}$$

Operaciones sobre cadenas

Las cadenas pueden concatenarse consigo mismas para conformar nuevas cadenas. Podemos definir la exponencia de cadenas como sigue:

Operador de exponente

Sea w una cadena, definimos el operador exponente como:

1. $w^0 := \epsilon$
2. $w^{i+1} := w^i \cdot w$

Por ejemplo, la cadena "ja" tiene los exponentes:

- $ja^1 = ja$
- $ja^2 = jaja$
- $ja^4 = jajajaja$

Operación suma y operación estrella

Dos operadores importantes serán el **operador suma** y el **Operador estrella** (o estrella de Kleen). Estos se pueden definir como:

- $w^+ := \{w^i : i > 0\}$
- Estrella de Kleen: $w^* := \{w^i : i \geq 0\}$

Podemos definir estos operadores para los alfabetos de la siguiente forma:

- $\Sigma^+ = \bigcup_{i>0} \Sigma^i$
- $\Sigma^* = \bigcup_{i\geq 0} \Sigma^i$

Conjunto de todas las cadenas

La operación de estrella nos permite definir el **conjunto de todas las cadenas** de un alfabeto:

Σ^*

Sea Σ un alfabeto. Al conjunto de todas las cadenas sobre Σ se le denota como Σ^* (sigma estrella).

La estructura $(\Sigma^*, \cdot, \epsilon)$ cumple las propiedades:

1. Elemento neutro: $\exists \epsilon \in \Sigma^*$ tal que $\epsilon \cdot w = w \cdot \epsilon = w$, $\forall w \in \Sigma^*$.
2. Asociatividad: $w_1 \cdot (w_2 \cdot w_3) = (w_1 \cdot w_2) \cdot w_3$

Esta estructura algebraica se conoce como **monoide** (Lambek, 2008). Es una estructura **no conmutativa**, lo que será relevante para el lenguaje natural.

Lenguaje

Es difícil determinar cuando hablamos de un lenguaje natural. En la teoría de los lenguajes formales, se define un lenguaje de forma sencilla:

Lenguaje

Un lenguaje sobre un alfabeto Σ es un subconjunto $L \subseteq \Sigma^*$.

Un lenguaje se definirá a partir de las cadenas que lo conformen. En lenguas como el español, podemos decir que las palabras, oraciones y discursos del español componen el lenguaje.

El problema fundamental al que no enfrentamos es como **determinar las cadenas** que pertenecen a un lenguaje de interés (español, inglés, náhuatl,...)

Operaciones sobre lenguajes

Al igual que sobre las cadenas, podemos definir algunas operaciones sobre los lenguajes:

Operaciones sobre lenguajes

Sean L y M dos lenguajes sobre un alfabeto Σ, entonces, se pueden realizar las siguientes operaciones:

1. $L \cdot M := \{x \cdot y | x \in L, y \in M\}$
2. $L^0 := \epsilon$
3. $L^{n+1} := L^n \cdot L$
4. $L^* := \bigcup_{n=0}^{\infty} L^n$ — Cerradura de Kleene
5. $L^+ := \bigcup_{n=1}^{\infty} L^n$
6. $L/M := \{y \in \Sigma^* | \exists x \in M \text{ tal que } y \cdot x \in L\}$ — Prefijos
7. $L \setminus M := \{y \in \Sigma^* | \exists x \text{ tal que } M \ni x \cdot y \in L\}$ — Sufijos

Clases de lenguajes

Clase de lenguajes

Sea Σ un alfabeto, y $\mathcal{L} = \{L \subseteq \Sigma^* || L| < \infty\}$ un conjunto de subconjuntos finitos de Σ^*. Entonces $\mathcal{L}$ es una clase de lenguajes.

Una propiedad relevante para las clases de lenguaje es que al operar sobre ellos el resultado siga perteneciendo a la misma clase de lenguajes. Esto define una estructura en las clases de lenguaje.

Cerrado bajo operación

Una clase de lenguajes, $\mathcal{L}$, es cerrada bajo una operación '$\circ$' si, dados cualquiera $L_1, L_2 \in \mathcal{L}$, se tiene que $L_1 \circ L_2 \in \mathcal{L}$.

Clases de lenguajes

Se distinguen cuatro clases de lenguajes principales que cumplen con una jerarquía:

1. **Tipo 0** o Lenguajes Recursivamente Numerables
2. **Tipo 1** o Lenguajes Dependientes del Contexto
3. **Tipo 2** o Lenguajes Libres de Contexto
4. **Tipo 3** o Lenguajes regulares

La jerarquía implica que: Tipo 3 $\subset$ Tipo 2 $\subset$ Tipo 1 $\subset$ Tipo 0

Para definir estos lenguajes, lo haremos a partir de las **gramáticas que los generan**.

Gramática

El concepto de **gramática** refiere a las **reglas** que regulan la **estructura** de una lengua.

La gramática sirve como una **construcción teórica** que describe y explica la lengua en todos sus **niveles**.

Su componente esencial son las **reglas**, las cuáles describen las relaciones de las unidades lingüísticas:

> ***R4.6*** *En la morfología del verbo se distinguen dos elementos constitutivos: la (también llamada* RADICAL*) , que expresa su significado tal como lo describe el diccionario, y un conjunto de* MORFEMAS FLEXIVOS *a los que corresponden dos funciones: establecer la concordancia de número y persona con el sujeto gramatical y expresar las nociones de 'modo', 'tiempo' y 'aspecto' que corresponden al evento.*

Reglas

Las **reglas** conforman una estructura formal que está definido por los sistemas de semi-Thue, también llamados **reglas de reescritura de cadenas**.

Sistema semi-Thue

Sea Σ^* un conjunto de cadenas sobre el alfabeto Σ; un sistema semi-Thue es una tupla $T = (\Sigma, R)$, donde $R \subseteq \Sigma^* \times \Sigma^*$ Cada elemento $(x, y) \in R$ generalmente se escribe como $x \mapsto y$.

Los sistemas de semi-Thue definirán las reglas de la gramática formal, que determinarán la estructura de las cadenas de una lengua.

Si la relación $x \mapsto y$ implica $y \mapsto x$, se dice que es un sistema de Thue.

Gramáticas

Gramática

Sea Δ un alfabeto (que llamaremos no terminal), $S \in \Delta$ el símbolo inicial, Σ otro alfabeto (llamado terminal) y R una relación tal que $R \subseteq (\Delta \cup \Sigma) \times (\Sigma \cup \Delta)$ conforma un sistema semi-Thue sobre $\Delta \cup \Sigma$. Una gramática es una 4-tupla, $G = (S, \Delta, \Sigma, R)$, tal que $\Delta \cap \Sigma = \emptyset$ y $\Delta \neq \emptyset \neq \Sigma$.

Si la gramática $G = (S, \Delta, \Sigma, R)$ genera a la cadena $w \in \Sigma^*$, se denotará como $G \to w$.

Lenguaje generado por una gramática

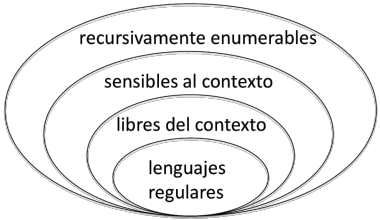
Al lenguaje generado por G se le define como sigue:

$$L(G) := \{x \in \Sigma^* : G \to x\}$$

Gramáticas y clases de lenguajes

Al igual que con los lenguajes, las gramáticas pueden ser clasificados en 4 tipos:

1. **Tipo 0:** Gramáticas Recursivamente Numerables
2. **Tipo 1:** Gramáticas Dependientes del Contexto
3. **Tipo 2:** Gramáticas Libres de Contexto
4. **Tipo 3:** Gramáticas regulares



Lenguaje tipo *i*

Un lenguaje de Tipo *i*, $i \in \{0, 1, 2, 3\}$ se define como un lenguaje generado por una gramática de Tipo *i*:

$$L_i(G) = \{w : G \to w, G \text{ es de Tipo i}\}$$

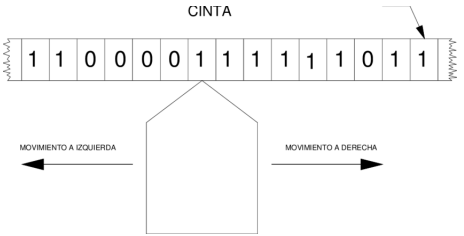
Gramáticas recursivamente numerables

En general, toda gramática (computable) se considera recursivamente numerable o de Tipo 0.

Las gramáticas de Tipo 0 son computables; los lenguajes de este tipo, entonces, pueden definirse como:

Lenguajes recursivamente numerables

Sea $L \subseteq \Sigma^*$ un lenguaje. L es recursivamente numerable si existe una función computable $f: \{0,1\}^* \to \Sigma^*$ tal que $L = f(\{0,1\}^*)$.



Gramáticas sensibles al contexto

Las **gramáticas sensibles al contexto** son aquellas cuyas reglas son del tipo:

$$c_1 X c_2 \rightarrow c_1 y c_2$$

El par de cadenas $c_1, c_2 \in \Sigma^*$ representa el **contexto**. Una forma común de denotar las reglas es:

$$X \rightarrow y / c_1 \_ c_2$$

Un ejemplo típico de lenguaje sensible al contexto es:

$$L = \{a^n b^n c^n : n > 0\}$$

Gramáticas sensibles al contexto

En PLN las gramáticas sensibles al contexto tienen algunas aplicaciones. Por ejemplo, pueden servir para definir **reglas ortográficas**.

Por ejemplo, sabemos que "'n"' no precede a "'p"' en español. Por lo que podemos proponer:

$$np \rightarrow mp$$

O bien:

$$n \rightarrow m/\_p$$

Al aplicar esta regla sensible al contexto tendremos casos como:

$$inposible \rightarrow imposible$$

Lenguajes regulares

Lenguaje regular

Los lenguajes regulares pueden pensarse como los lenguajes más simples. Se pueden definir recursivamente a partir de operaciones sobre lenguajes:

Lenguaje regular

Un lenguaje regular sobre un alfabeto Σ se define como:

1. $L = \emptyset$ es un lenguaje regular.
2. $L = \{\epsilon\}$ es un lenguaje regular.
3. $L = \{a : a \in \Sigma^*\}$ es un lenguaje regular.
4. Si L_1 y L_2 son lenguajes regulares, $L_1 \cdot L_2$, $L_1 \cup L_2$ y L_1^* son lenguajes regulares.

Un lenguaje regular puede pensarse como aquel lenguaje que **permite estas operaciones**.

Gramática regular

Un lenguaje regular se puede definir a partir de las reglas de la gramática que las genera, una **gramática regular** contiene reglas de la forma:

$$X \to a \cdot Y$$

Donde $a \in \Sigma \cup \{\epsilon\}$, y $Y \in \Delta \cup \{\epsilon\}$. Por ejemplo (. denota cualquier símbolo):

$$S \to . \cdot X$$
$$X \to a \cdot Y$$
$$Y \to t \cdot Z$$
$$Z \to o|a$$

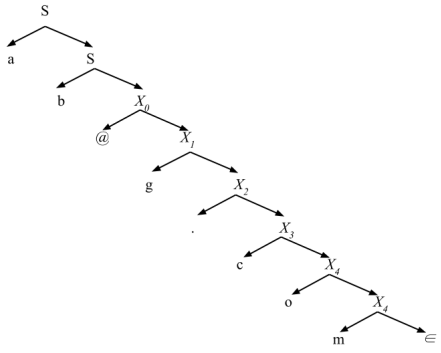
Genera un lenguaje regular: ¿Cuáles son las cadenas que contiene el lenguaje generado por esta gramática?

Parser de lenguajes regulares

Si se quiere construir una gramática que exprese el lenguaje de los correos electrónicos, con esta se podrán **parsear** expresiones. Consideremos la gramática:

$$
\begin{aligned}
S &\to .\cdot S|.\cdot X_0 \\
X_0 &\to @\cdot X_1 \\
X_1 &\to .\cdot X_1|.\cdot X_2 \\
X_2 &\to /.\cdot X_3 \\
X_3 &\to c\cdot X_4 \\
X_4 &\to o\cdot X_5 \\
X_5 &\to m
\end{aligned}
$$

El parseo determina un **árbol** que la describe la expresión:



Árbol de la expresión **ab@g.com**

Autómatas finitos

Ligado a los lenguajes regulares, encontramos los autómatas finitos, los cuales permiten procesar los lenguajes regulares.

Autómata de estados finitos

Un autómata de estados finitos es una 5-tupla $A = (Q, \Sigma, q_0, F, \delta)$, tal que $Q = \{q_0, q_1, ..., q_T\}$ es un conjunto de estados; Σ es el alfabeto de entrada; $q_0 \in Q$ es el estado inicial; $F \subseteq Q$ es el conjunto de estados finales; y $\delta : Q \times \Sigma \to Q$ es la función de transición.

Nota: La transiciones δ puede ser dadas por una **función parcial**; esto es:

$$\delta : Q \times \Sigma \to \mathcal{P}(Q)$$

Esto marca la diferencia entre autómatas determinísticos y no determinísticos.

Lenguaje definido por autómata

El autómata finito, entonces, determina un lenguaje:

Lenguaje definido por un autómata

El *lenguaje de* A, denotado $L(A)$, es el conjunto de todas las cadenas que el autómata acepta; es decir:

$$L(A) = \{w \in \Sigma^* | \exists q_f \in F \text{ tal que } (q_0, w, q_f) \in \hat{\delta}\}$$

Dentro de las transiciones del autómata tenemos transiciones nulas o ϵ:

Transiciones ϵ

Un autómatas de estados finitos con *transiciones* ϵ es un autómata, $A = (Q, \Sigma, q_0, F, \delta)$, tal que $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to Q$.

Equivalencia de atuómatas

Puede ser que dos autómatas estén definiendo el mismo lenguaje, entonces se dice que son **equivalentes**:

Equivalencia entre autómatas

Se dice que dos autómatas de estados finitos, A_1 y A_2 son equivalentes si $L(A_1) = L(A_2)$.

En general, todo autómata finito no determinístico con T estados puede convertirse en un autómata finito determinístico con a lo mucho 2^T estados.

Función de transición

Dado un símbolo $c \in \Sigma$ y un estado $q, q' \in Q$, las función de transición puede expresarse como:

- Una relación de elementos (q, c, q') o bien $(q, c, \{q_i : q_i \in Q\})$
- Una función (o función parcial): $\delta(q, c) = q'$ o bien $\delta(q, c) = \{q_i : q_i \in Q\})$.
- Una matriz de transiciones

$$(\delta_{q,c}) = q'$$

En este caso, tenemos una tabla de la forma:

	c_1	c_2	$\cdots$	c_n
q_0	q'	0	0	0
q_1	0	$\{q', q''\}$	0	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
q_T :	0	0	0	0

Los 0 indican que no hay transición y **:** indica estado final.

Tipos de autómatas

Es común hablar de dos tipos de autómatas según su uso o no de transiciones nulas:

Autómata finito determinístico

Un autómata $A = (Q, q_0, \Sigma, \delta, F)$ es determinístico si $\delta : Q \times \Sigma \to Q$ (no tiene transiciones ϵ).

Autómata finito no-determinístico

Un autómata finito es no-determinístico si no es determinístico. En particular, la función de transición es de la forma $\delta : Q \times \Sigma \to \mathcal{P}(Q)$.

Un autómata determinístico puede llevar a varios estados desde un mismo estado y con un mismo símbolo.

Cerradura transitivo-reflexiva

Una forma común de simplificar los autómatas es con la cerradura transitiva relfexiva:

Cerradura transitiva reflexiva de la función de transición

Dado un autómata de estados finitos $A = (Q, \Sigma, q_0, F, \delta)$ la cerradura transitiva reflexiva $\hat{\delta}$ se define de la siguiente manera:

- Para todo estado $q \in Q$, se cumple que $(q, \epsilon, q) \in \hat{\delta}$
- Si $(q, a, q') \in \hat{\delta}$ y $(q', b, q'') \in \hat{\delta}$ entonces $(q, a \cdot b, q'') \in \hat{\delta}$, para todos $a, b \in \Sigma$.

Se dice que w es *aceptado* por A si existe un estado final $q_f \in F$ tal que $(q_0, w, q_f) \in \hat{\delta}$, donde $w \in \Sigma^*$ es una cadena.

De esta forma podemos escribir autómatas más simples. No haremos distinción entre δ y $\hat{\delta}$.

Ejemplo de autómata

Consideremos el siguiente lenguaje regular:

$$L = \{ama, amo, aman, amas, amamos\}$$

Podemos construir la gramática regular como sigue:

$$X_0 \rightarrow am \cdot X_1$$
$$X_1 \rightarrow a \cdot X_2 | a | o$$
$$X_2 \rightarrow n | s | mos$$

Los símbolos no-terminales son $\Delta = \{X_0, X_1, X_2, \epsilon\}$, el símbolo inicial es X_0.

¿Cuáles serían los árboles para: 'ama', 'amamos',...?

Ejemplo de autómata

Podemos definir una función que transforme cada símbolo no-terminal en un estado. Agregamos estados finales cada vez que tenemos un terminal.

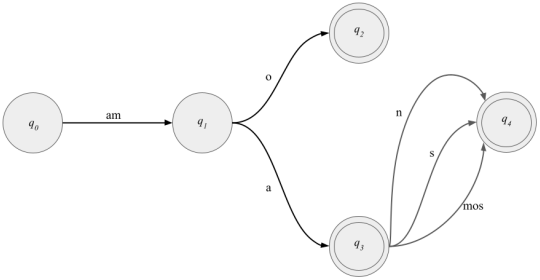
Definimos la función de transición como sigue:

$$\begin{aligned}\delta(q_0, am) &= q_1\\ \delta(q_1, o) &= q_2\\ \delta(q_1, a) &= q_3\\ \delta(q_3, n) &= q_4\\ \delta(q_3, s) &= q_4\\ \delta(q_3, mos) &= q_4\end{aligned}$$

Los estados finales están dados por $F = \{q_2, q_3, q_4\}$.

Ejemplo de autómata

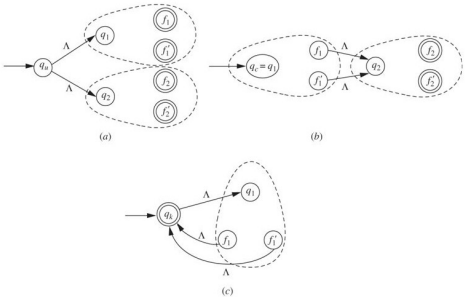
	am	o	a	n	s	mos
q_0	q_1	0	0	0	0	0
q_1	0	q_2	q_3	0	0	0
q_2 :	0	0	0	0	0	0
q_3 :	0	0	0	q_4	q_4	q_4
q_4 :	0	0	0	0	0	0



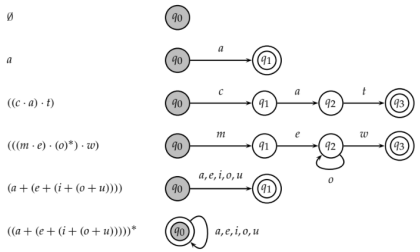
Operaciones de autómatas

Algunas propiedades que se pueden ver en los autómatas de estados finitos con transiciones ϵ son las siguientes:

1. Concatenación. A partir de una transición ϵ.
2. Kleene *. Conectar el final con el inicio del autómata a partir de transiciones ϵ.
3. Unión. Suma de los lenguajes definidos por los autómatas.



Autómatas y lenguajes regulares



De gran importancia son los siguientes resultados, pues relacionan los atuómatas finitos con los lenguajes regulares (Clark et al., 20139):

Teorema de Kleen

Para todo lenguaje regular, L, existe un autómata de estados finitos, A, que lo define. Es decir, todo lenguaje regular es un lenguaje generado por un autómata $L = L(A)$.

Corolario

El lenguaje $L(A)$ definido por un autómata es un lenguaje regular.

Expresiones regulares

Las **expresiones regulares** son expresiones en cadenas que definen de manera abstracta un patrón. Estas definen a los lenguajes regulares.

RegEx	Explicación
.	Cualquier caracter
$\widehat{}$	El comienzon de una cadena
$	El final de una cadena
$\backslash d$	Un número entre 0 y 9

A partir de las expresiones regulares se pueden construir autómatas que sirvan para reconocer los patrones expresados por la regex.

Transductor finito

Los autómatas finitos no permiten generar ni parsear lenguajes regulares. Por tanto, necesitamos definir una gramática para los lenguajes regulares:

Transductores de estados finitos

Sea $Q = \{q_0, q_1, ..., q_T\}$ un conjunto de estados; Σ un alfabeto de entrada; $q_0 \in Q$ el estado inicial; y $F \subseteq Q$ un conjunto de estados finales. Un transductor de estados finitos es una 6-tupla $T = \{Q, \Sigma, \Delta, q_0, F, \sigma\}$ tal que: Δ es un alfabeto de de entrada (no terminal) y Σ uno de salida (terminal) y $\sigma : Q \times \Delta \times \Sigma \rightarrow Q$ es una función de transducción.

Ejemplo: transductor finito

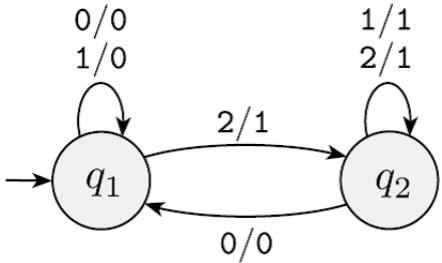
Consíderese el siguiente transductor finito:

- Conjunto de estados: $Q = q_1, q_2$
- Alfabeto de entrada: $\Delta = 0, 1, 2$
- Alfabeto de salida: $\Sigma = \{0, 1\}$
- Inicial y Estados finales: $q_1 \in Q$, $F = \{q_2\}$
- Función de transición:

$$\sigma(q_1, 0, 0) = q_1$$
$$\sigma(q_1, 1, 0) = q_1$$
$$\sigma(q_1, 2, 1) = q_2$$
$$\sigma(q_2, 1, 1) = q_2$$
$$\sigma(q_2, 2, 1) = q_2$$
$$\sigma(q_2, 0, 0) = q_1$$

Ejemplo: transductor finito

Gráficamente, este transductor puede representarse como:



Algunas transducciones que este transductor finito puede hacer son:

$$022 \mapsto 011$$

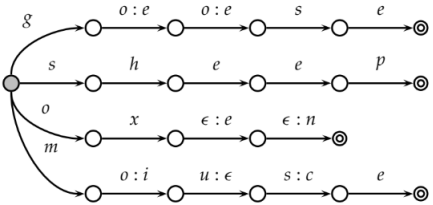
$$1221 \mapsto 0111$$

¿Cuál es el lenguaje de entrada y cuál el de salida?

Transiciones nulas

Se pueden dar transiciones **epsilon**, ϵ, en los transductores ya sea como parte de la entrada o la salda. Es decir, tenemos los casos:

- Inserta un símbolo: $\sigma(q, \epsilon, w) = q'$, $w \in \Sigma$.
- Elimina un símbolo: $\sigma(q, a, \epsilon) = q'$, $a \in \Delta$.
- Transición nula: $\sigma(q, \epsilon, \epsilon) = q'$.



Cerradura transitivo reflexiva

Cerradura transitivo-reflexiva en transductores

Sea $T = (Q, \Sigma, \Delta, q_0, F, \sigma)$ un transductor finito. La cerradura transitivo-reflexiva $\hat{\sigma} : Q \times \Delta^* \times \Sigma^* \to Q$ se define como:

1. Para todo $q \in Q$, $(q, \epsilon, \epsilon, q) \in \hat{\sigma}$.
2. Si $(q, a, w_1, q') \in \hat{\sigma}$ y $(q', b, w_2, q'') \in \hat{\sigma}$, entonces $(q, a \cdot b, w_1 \cdot w_2, q'') \in \hat{\sigma}$.

Un par (x, y) tal que $x \in \Delta^*$ y $x \in \Sigma^*$ es generado por T si $\hat{\sigma}(q_0, x, y) = q_f$ para algún $q_f \in F$.

Si un par (x, y) es generado por el transductor T, podemos escribir xTy. Decimos que la **relación definida por el transductor** es el conjunto de todos los pares que genera.

Operaciones sobre transductores finitos

Al igual que con los autómatas de estados finitos, los transductores pueden operarse a partir de:

- **Unión:** Dados T, S transductores, su unión $T \cup S$ agrega un estado inicial q' y transiciones $(q', \epsilon, \epsilon, q_T)$ y $(q', \epsilon, \epsilon, q_S)$.
- **Concatenación:** Dados T, S, con $\{q_T\}$ estados finales de T, y q_S estado inicial de S, su concatenación $T \cdot S$ agrega la transición(es) $(q_T, \epsilon, \epsilon, q_S)$.
- **Estrella de Kleen:** Dado un transductor T, con estado inicial q_0 y final q_f, la estrella de Kleen sobre este transductor, T^* agrega las transiciones $(q_0, \epsilon, \epsilon, q_f)$ y $(q_f, \epsilon, \epsilon, q_0)$.
- **Proyección:** La proyección π_Δ de T se da como: $\pi_\Delta(T) = \{x \in \Delta^* : \exists y, xTy\}$. De manera similar definimos π_Σ.

Análisis morfológico

Análisis morfológico

La morfología de las lenguas suele mostrar patrones recurrentes que hacen factible la aplicación de gramáticas regulares.

Algunos procesos del análisis morfológico son:

- Lematización. Llevar a la palabra a su lema.
- Parseo morfológico. Analizar las partes que componen la palabra y sus significados.
- Stemming. Truncar la palabra para reducir número de tipos.

Morfología

La **morfología** estudia la estructura interna de las palabras. Su objetivo en PLN es reducir el número de tipos al determinar cuàndo diferentes tokens pertenecen a una misma palabra.

Morfología

La morfología es el estudio de las co-variaciones sistemáticas en la forma y el significado de las palabras.

Por ejemplo:

	AMAR	TOMAR	SER
Presente	am-a	tom-a	es
Pasado	am-ó	tom-ó	fue
Futuro	am-ará	tom-ará	será

Morfología

Las unidades de análisis de la morfología son los **morfemas**. Estos se han definido como:

Morfema

Un morfema es la unidad mínima con significado en una producción lingüística.

Por ejemplo, la siguiente lista de palabras:

- niñ-a
- niñ-o
- niñ-a-s
- niñ-o-s

cuenta con los morfemas "'-a"' 'femenino', "'-o"' 'masculino', "'-s"' 'plural' y "'niñ-"' 'infante'.

Morfología

Sin embargo, existen morfemas que según su distribución pueden presentar diferentes formas:

- in-tocable
- im-personal
- i-nato

El objetivo de la morfología es describir la distribución de los morfemas: con qué otros morfemas pueden combinarse para producir palabras; qué formas toman; qué significados aportan.

Bases y afijos

Las palabras pueden dividirse en dos tipos de subcadenas:

Base

La base es el elemento de una palabra que es susceptible a un proceso morfológico. También se conoce como *stem*.

Afijo

Son los elementos morfológicos (subcadenas) que se concatenan a las bases para crear nuevas palabras a partir de ellas.

Afijo	**Base**	**Afijo**
im	posi	ble

Subcadenas y afijos

En la teoría de lenguajes formales tenemos conceptos que nos servirán para el análisis morfológico.

Subcadena

Sea $w \in \Sigma^*$ una cadena, $y \in \Sigma^*$ es una subcadena de w si $\exists x, z \in \Sigma^*$ tal que $w = x \cdot y \cdot z$.

Afijos

Sea $x, y \in \Sigma^*$. Se dice que y es un *prefijo* de x si $x = y \cdot w$, para algún $w \in \Sigma^*$. Se dice que y es un *sufijo* de x si $x = w \cdot y$. Los prefijos y los sufijos conforman los afijos.

Prefijo	**Base**	**Sufijo**
im	posi	ble

Contexto

Relacionado con los conceptos anteriores, tenemos:

Contexto

Un contexto es un par $N = (y, z)$ de cadenas. La *substitución de x dentro de N*, $N(x)$, es la cadena $y \cdot x \cdot z$.
Decimos que x ocurre en w en el contexto N si $w = N(x)$.

La idea del análisis computacional de la morfología es determinar la distribución de morfemas (prefijos, sufijos y bases) en los diferentes contextos en que pueden aparecer.

Parseo morfológico

El **parseo morfológico** es la determinación de las unidades que componen a la palabra y su representación lingüística.

Busca encontrar los morfemas en una palabra y los significados/funciones de éstos:

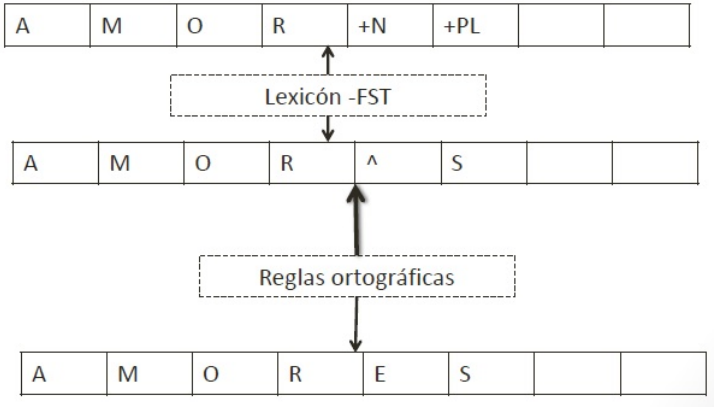
$$
\begin{aligned}
\textit{gatos} &\mapsto \textit{gat} - \textit{o}\{\text{MSC}\} - \textit{s}\{\text{PL}\} \\
\textit{gatas} &\mapsto \textit{gar} - \textit{a}\{\text{FEM}\} - \textit{s}\{\text{PL}\} \\
\textit{ventanas} &\mapsto \textit{ventana} - \textit{s}\{\text{PL}\} \\
\textit{flores} &\mapsto \textit{flor} - \textit{es}\{\text{PL}\}
\end{aligned}
$$

Parseo morfológico

Para realizar un parseo morfológico se requieren de los siguientes elementos (Jurafski y Martin, 2000):

- **Lexicón**. La lista de bases de palabra y afijos. Es decir L y los afijos $L/M \cup L \setminus M$.
- **Reglas morfotácticas**. Son las reglas (o la gramática) que determinarán el comportamiento morfológico de una palabra.
- **Reglas ortográficas**. Las reglas ortográficas determinan los cambios ortográficos (alomorfía) que se presenta en los parseos morfológicos.

Parseo morfológico



Ejemplo: parseo de plural

Supóngase que se cuenta con un lenguaje de sustantivos que cuentan con formas singulares y plurales:

$$L_{PL} = \{\text{gato, gatos, flor, flores, análisis}\}$$

El conjunto de bases está dado por:

$$M = \{\text{gato, flor, análisis}\}$$

Y el conjunto de sufijos de plural es:

$$L_{PL} \setminus M = \{-s, -es\}$$

Ejemplo: parseo de plural

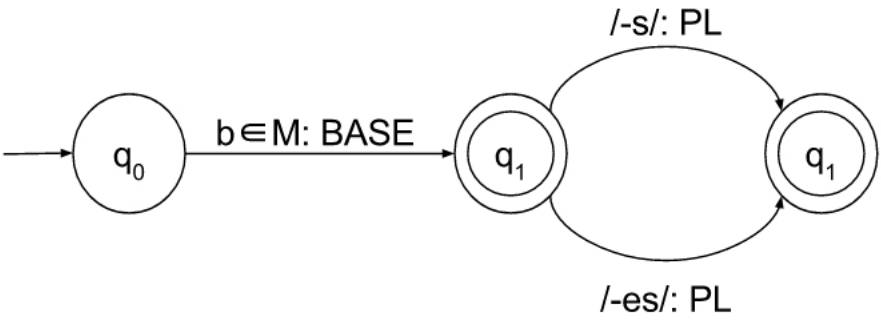
Necesitamos definir, entonces, un transductor finito que tome estos elementos y los transduzca en sus funciones:

- Alfabeto de entrada: $\Delta = M \cup L_{PL} \setminus M$
- Conjunto de estados: $Q = \{q_0, q_1, q_2\}$
- Conjunto de estados finales: $F = \{q_1, q_2\}$
- Alfabeto de salida: $\Sigma = \{\text{BASE}, \text{PL}\}$
- Función de transducción: Sea $b \in M$:

$$\sigma(q_0, b, BASE) = q_1$$
$$\sigma(q_1, s, PL) = q_2$$
$$\sigma(q_1, es, PL) = q_2$$

Ejemplo: parseo de plural

El transductor resultante se puede ver gráficamente como:



Ejemplo: parseo de plural

Se puede reducir la función de transducción al incorporar un módulo de **reglas ortográficas**:

1. Si existe la subcadena /s/ al final de la palabra, sustitúyase por /ss/.
2. Si existe la subcadena /es/, sustitúyase por /s/.

La nueva función de transducción será:

$$\sigma(q_0, b, BASE) = q_1$$
$$\sigma(q_1, s, PL) = q_2$$

Generación de morfología

Como proceso inverso al parseo, podemos hablar de la **generación morfológica**:

Generación morfológica

La generación morfológica busca generar formas de palabra a partir de una base y su descripción morfológica.

Buscamos un mapeo de la forma:

$$\text{gat} + \text{MSC} + \text{PL} \rightarrow \textit{gatos}$$

Generación morfológica

Para hacer un método de generación con un transductor finito, primero se debe determinar una **clase de palabra**. Nosotros tomamos los **sustantivos** del español.

Además debemos describir las palabras: cuál es la **estructura que tiene**:

$$\text{BASE} + (\text{DIM}) + \text{GEN} + \text{NUM}$$

DIM		it
GEN	FEM	a
	MSC	o
NUM	SG	ϵ
	PL	s

Generación morfológica

Un posible transductor finito que genere el lenguaje de plurales en español es:

- $\Delta = M \cup \{MSC, FEM, SG, PL\}$
- $\Sigma = M \cup \{o, a, s, \epsilon\}$
- $Q = \{q_0, q_1, q_2, q_3\}$, con q_0 inicial
- Reglas de transducción:

$$\sigma(q_0, b, b) = q_1$$
$$\sigma(q_1, FEM, a) = q_2$$
$$\sigma(q_1, MSC, o) = q_2$$
$$\sigma(q_2, SG, \epsilon) = q_3$$
$$\sigma(q_2, PL, s) = q_3$$

Stemming

El proceso de **stemming** o truncamiento consiste en reducir un token léxico a una supuesta base por medio de truncar los afijos correspondientes.
Esto es:

$$w \mapsto b \in M$$

Donde M es el conjunto de bases de una lengua ($w = x \cdot b \cdot y$).

El proceso de stemming es similar al parseo morfológico, pero mapea los afijos a elementos nulos ϵ.

Algoritmo de Porter

El **Algoritmo de Porter** es un algoritmo de stemming basado en transductores finitos enfocado a **sufijos**.
Tiene varias fases; la primera de ellas es buscar **regiones** que se aproximen a las sílabas en una lengua:

- R1: después de la primera consonante que sigue a una vocal, o la región nula al final de la palabra si no existe dicha vocal.
- R2: después de la primera consonante que sigue a una vocal dentro de R1, o la región nula al final de la palabra si no existe dicha vocal.
- RV: si la segunda letra es una consonante, después de la siguiente vocal subsecuente. Si las dos primeras letras son vocales, después de la consonante subsiguiente. En otro caso, después de la tercera letra.

Algoritmo de Porter

Un ejemplo de una regla del algoritmo de Porter para el español es:

1. Busca las subcadenas "'iéndo"', "'ándo"', "'ár"', "'ér"', "'ír"' (con o sin acento) dentro de la región RV.
2. Si el paso anterior se cumple, y a las subcadenas definidas en el paso anterior le siguen los sufijos "'me"', "'se"', "'sela"', "'selo"', "'selas"', "'selos"', "'la"', "'lo"', "'le"', "'las"',"'los"', "'les"' o "'nos"', eliminar estos sufijos.

El algoritmo para español puede encontrarse en:

http://snowball.tartarus.org/algorithms/spanish/stemmer.html

Lematización

En la **lematización** un token es llevado a su lema.

Para hacer esto, se utiliza un diccionario que relaciona los tipos con los lemas. Es común que se haga un proceso de stemming antes para reducir el número de entradas del diccionario.

Token	Stem	Lema
niñita	niñ	niño
niños	niñ	niño
podemos	pod	poder
puedo	pued	poder

Lenguajes libres de contexto

Gramáticas Libres de Contexto

En una gramática libre de contexto, dado un **símbolo no-terminal** $X \in \Delta$, se obtienen **cadenas terminales y/o no terminales** $A \in (\Sigma \cup \Delta)^*$:

$$X \mapsto A$$

Gramática libre de contexto

Una gramática libre de contexto, *CFG*, es una cuatro-tupla $CFG = (\Delta, \Sigma, d_0, R)$, donde $\Delta = \{d_0, ..., d_n\}$ es un conjunto de símbolos no terminales, Σ es un conjunto de símbolos terminales, $d_0 \in \Delta$ es el símbolo de inicio y $R : \Delta \to \Delta \cup \Sigma$ es un sistema de reglas.

Una **gramática regular** es una gramática libre de contexto ya que $a \cdot Y \in (\Sigma \cup \Delta)^*$.

Gramáticas libres de contexto

Cerradura transitiva reflexiva para una CFG

Dada una gramática libre de contexto $CFG = (\Delta, \Sigma, d_0, R)$ donde las reglas $(d, w) \in R$ se denotan como $d \mapsto w$ con $d \in \Delta$ y $w \in \Delta \cup \Sigma$, la cerradura transitiva-reflexiva (denotada $\mapsto^+$) cumple:

1. Si $d_1 \mapsto d_2$ y $d_2 \mapsto w$ entonces $d_1 \mapsto^+ w$. Esta es la propiedad de transitividad (claramente d_2 debe ser no terminal).
2. Sea $n \geq 2$, y $d_1 \neq d_{n+1}$, entonces, $d_1 \mapsto d_2 \mapsto \cdots \mapsto d_n \mapsto d_{n+1}$

Propiedades de las CFG

Propiedades de una CFG

Sea $G = (\Delta, \Sigma, d_0, R)$, entonces, se tiene que el sistema de reglas R cumple:

1. $\forall d \in \Delta$, existen símbolos del tipo $d_0 \rightarrow^* \alpha \cdot d \cdot \beta$ para $\alpha, \beta \in \Delta \cup \Sigma$.
2. $\forall d \in \Delta$ existe al menos un elemento $w \in \Sigma$ tal que $d \mapsto^+ w$.
3. $\forall d \in \Delta$ es imposible tener una regla del tipo $d \mapsto^+ \epsilon$.
4. $\forall d \in \Delta$ y $\forall n \geq 2$ no es posible que $d \mapsto d_1 \mapsto \cdots \mapsto d_n \mapsto d$.

Lenguaje Libre de Contexto

Una CFG así definida, entonces, cumple lo siguiente (Kracht, 2011):

- Es una gramática.
- Los símbolos a la izquierda son siempre no-terminales.
- Tiene una estructura jerárquica (un árbol).
- Sólo termina cuando todas las ramas del árbol son símbolos terminales.

Un **lenguaje libre de contexto** $L(CFG) \subseteq \Sigma^*$ es el lenguaje generado por una CFG (Δ, Σ, S, R):

$$L(CFG) = \{w \in \Sigma^* : S \mapsto^+ w\}$$

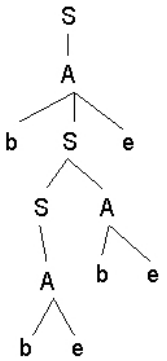
Ejemplo: CFG

Considérese la siguiente gramática libre de contexto $CFG = \{\Delta, \Sigma, d_0, R\}$, tal que:

- Alfabeto no terminal: $\Delta = \{S, A\}$
- Alfabeto terminal: $\Sigma = \{b, e\}$
- Símbolo inicial: $d_0 = S$
- Sistema de reglas R:

$$S \mapsto SA$$
$$S \mapsto A$$
$$A \mapsto bSe$$
$$A \mapsto be$$

El lenguaje generado es $L(CFG) = \{bbebee, be, bbee, ...\}$. El árbol determinado por *bbebee* es:



Forma estándar

Forma estándar

Una gramática libre de contexto $CFG = (\Sigma, \Delta, S, R)$ está en forma estándar si todas las reglas son de la forma $X \to Y$ o $X \to x$, donde X es no terminal, $Y \in \Delta^*$ y $x \in \Sigma^*$.

Proposición

Para toda CFG existe una CFG en forma estándar que genera el mismo lenguaje.

La gramática del ejemplo anterior puede convertirse en forma estándar como:

$$S \mapsto SA|A$$
$$A \mapsto BSE|be$$
$$B \mapsto b$$
$$E \mapsto e$$

Forma normal de Chomsky

Forma normal de Chomsky

Se dice que una gramática libre de contexto $CFG = (\Sigma, \Delta, S, R)$ está en forma normal de Chomsky si todas las reglas son de la forma $X \to Y_0 \cdot Y_1$ o $X \to x$, donde X es no terminal, $Y_0, Y_1 \in \Delta$ y $x \in \Sigma$.

Proposición

Para toda CFG existe una CFG en forma normal de Chomsky equivalente. Además toda gramática en forma normal de Chosmky es una CFG.

La gramática del ejemplo anterior puede convertirse en forma estándar como:

$$S \mapsto SA|BX|BE$$
$$A \mapsto BX|BE$$
$$X \mapsto SE$$
$$B \mapsto b$$
$$E \mapsto e$$

Forma normal de Chomsky

Para obtener una gramática en forma normal de Chomsky a aprtir de una gramática cualquiera, se siguen los siguientes pasos:

1. Sustituir $X \to Y_0 \cdot ... \cdot x \cdot ... \cdot T_n$ por $X \to Y_0 \cdot ... \cdot Y_x \cdot ... \cdot T_n$ y agregar $Y_x \to x$.
2. Eliminar $X \to Y_0$ y $Y_0 \to Y_1 \cdot ... \cdot Y_n$, y agregar $X \to Y_1 \cdot ... \cdot Y_n$
3. Eliminar $X \to \epsilon$
4. Sustituir las reglas de la forma $X \to Y_0 \cdot Y_1 \cdot ... \cdot Y_n$ por reglas:

$$X \to Y_0 \cdot Z_0$$
$$Z_0 \to Y_1 \cdot Z_1$$
$$...$$
$$Z_{n-2} \to Y_{n-1} \cdot Y_n$$

Forma normal de Greibach

Forma de Greibach

Sea $G = \{\Sigma, \Delta, S, R\}$ una gramática y $X \in \Delta$ un símbolo no terminal. Se dice que G está en forma (normal) de Greibach si toda regla en R es de la forma $X \to x \cdot Y_0 \cdot Y_1 \cdot ... \cdot Y_n$, con $x \in \Sigma$ y $Y_0 \cdot Y_1 \cdot ... \cdot Y_n \in \Delta^*$.

Proposición

Para toda CFG existe una CFG en forma de Greibach equivalente.

Considérese la gramática en forma estándar:

$$S \to X \cdot S \cdot Y | X \cdot Y$$
$$X \to a$$
$$Y \to b$$

Su forma normal de Greibach puede ser:

$$S \to a \cdot S \cdot X | a \cdot X$$
$$X \to b$$

Autómatas con pila

Autómata con pila

Un autómata con pila sobre el alfabeto Σ es una 7-tupla $A = (\Sigma, \Gamma, \gamma_0, Q, q_0, F, \delta)$, donde Γ es la pila, $\gamma_0 \in \Gamma$ un símbolo que indica inicio de la pila, Q un conjunto de estados, $q_0 \in Q$ el estado inicial, $F \subseteq Q$ un conjunto de estados finales y $\delta : Q \times \Sigma \cup \{\epsilon\} \times \Gamma \to Q \times \Gamma^*$ una función de transición.

Un autómata con pila realiza dos operaciones básicas sobre la pila: **push** y **pop**.

El **lenguaje** definido por el autómata es:

$$L(A) = \{w \in \Sigma^* : \exists q_f \in F, \hat{\delta}(q_0, w, \gamma_0) = (q_f, \gamma_0)\}$$

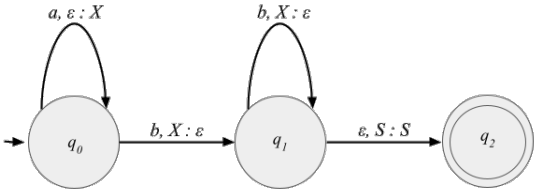
Lenguaje libre de contexto

Un lenguaje libre de contexto L es aquel que es aceptado por un autómata con pila A.

Ejemplo: Autómata con pila

Sea $A = (\Sigma, \Gamma, S, Q, q_0, F, \delta)$, donde $\Sigma = \{a, b\}$, $\Gamma = \{S, X\}$, donde S será el inicio de pila, $Q = \{q_0, q_1, q_2\}$ y $F = \{q_2\}$. δ definido como:

$$\delta(q_0, a, S) = (q_0, X \cdot S)$$
$$\delta(q_0, a, X) = (q_0, X \cdot X)$$
$$\delta(q_0, b, X) = (q_1, \epsilon)$$
$$\delta(q_1, b, X) = (q_1, \epsilon)$$
$$\delta(q_1, \epsilon, S) = (q_2, S)$$



Análisis sintáctico

Sintaxis

La **sintaxis** es el estudio de la estructura de las oraciones.

¿Qué elementos estructuran las oraciones?

Consideremos las siguientes oraciones:

- El automóvil es rojo.
- El viejo automóvil de Juan es rojo.
- El viejo automóvil rojo se estrelló contra un muro en la madrugada.

¿Cuál es la estructura de estas oraciones? ¿Hay otros elementos constitutivos además de las palabras?

Sujeto y predicado

Lo primero que se puede observar es que existe un **sujeto** y un **predicado**.

Sujeto: Es quien realiza la acción del verbo.

Predicado: Es la acción que toma como argumento al sujeto.

Podemos representar al predicado como una función $\phi : \Sigma^* \to \Sigma^*$ [**?**]. Por ejemplo:

$$\phi(\cdot) := \cdot \text{ es rojo}$$

De tal forma que:

$$\phi(\text{el automóvil}) = \text{el automóvil es rojo}$$

Oraciones complejas

Una oración puede ser más compleja. La tercera oración puede ser vista como:

$$\phi(x) = X \text{ se estrelló contra la pared en la madrugada}$$

Pero también "'la pared"' y "'la madrugada"' pueden ser argumentos. Tal que:

$$\phi(x_1, x_2, x_3) = x_1 \text{ se estrelló contra } x_2 \text{ en } x_3$$

De tal forma que podemos construir una nueva oración como:

$$\phi(\text{el ave}, \text{el árbol}, \text{la noche}) = \text{el ave se estrelló contra el árbol en la noche}$$

Constituyentes oracionales

Por su función en las oraciones, podemos analizar los elementos de éstas acercándonos cada vez a las palabras:

El viejo automóvil rojo	se estrelló	contra	un muro	en	la madrugada
El viejo automóvil rojo	se estrelló	contra	un muro	en	la madrugada
El viejo automóvil rojo	se estrelló	contra	un muro	en	la madrugada
El viejo automóvil rojo	se estrelló	contra	un muro	en	la madrugada

⋮

De manera general, podemos hablar de **constituyentes** de la oración.

Frases

Los constituyentes tienen una estructura particular. Por ejemplo, pueden contener un verbos, un sustantivo, una preposición.

En particular, hablaremos de **frases** o sintagmas:

Frase

Una frase o sintagma es un grupo de elementos léxicos (o bien palabras) que forman sub-constituyentes de una oración.

Cada frase tiene un **núcleo** que es la palabra central, que aporta el significado esencial de la frase.

Tipos de frases

Las frases se distinguen según su núcleo. Algunos tipos de frase, aunque no todas, pueden ser:

Frase nominal (FN). La frase nominal se compone de un sustantivo como núcleo. Por ejemplo, "'el gato"', "'el gato viejo"', "'los perros negros"'.

Frase verbal (FV). Es la frase cuyo núcleo es un verbo. Ejemplos: "'corre"', "'comí carne"' "'persiguió a los gatos"'.

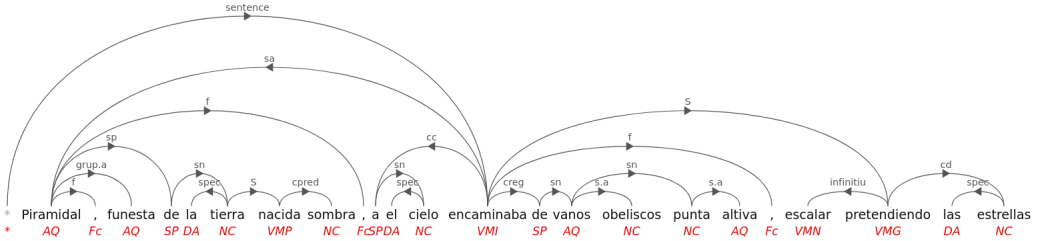
Frase preposicional (FP). Es la frase cuyo núcleo es una preposición. Por ejemplo: "'del gato"', "'con el perro"'.

Parseo sintáctico: dependencias

Distinguimos dos tipos básicos de parseo **según las relaciones** de los elementos en la oración. El primero de ellos es el parseo de dependencias:

Parseo de dependencias

El parseo de dependencias busca identificar las dependencias que se dan entre las palabras de una oración. Es decir, para cada token, identifica de cuál otro token depende.

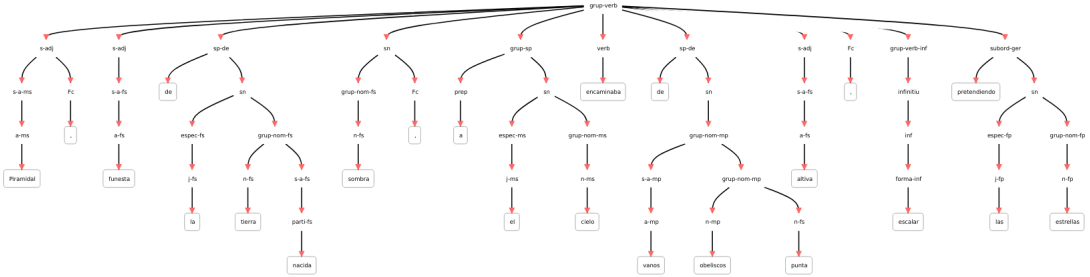


Parseo sintáctico: constituyentes

El otro tipo de parsing es el llamado parseo de constituyentes

Parseo de constituyentes

El parseo de constituyentes busca identificar los constituyentes de una oración, tales como las frases y los elementos que las componen.

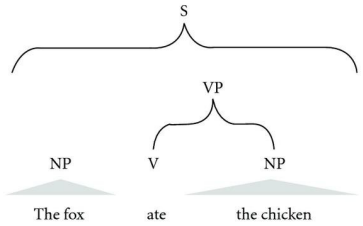


Parseo de constituyentes

Dentro del parseo de constituyentes, dependiendo de la **profundidad** con que se parseen los elementos, tenemos dos tipos de parsing:

(Full) parsing: Se parsea una oración señalando la estructura sintáctica de dicha oración.

Shallow parsing: Se parsea una parte limitada de la información sintáctica (*chunks*). También llamado *chunking* (Abney, 1991).



Análisis sintáctico

Considérese el siguiente corpus que contiene un lenguaje a analizar:

- El gato come sopa
- El perro come huesos
- El gato come del plato

Las estructuras sintácticas que definen estas oraciones son:

- $([\text{El}_{Art}\ \text{gato}_{NN}]_{FN}\ [\text{come}_{V}\ [\text{sopa}_{NN}]_{FN}]_{FV})_{O}$
- $([\text{El}_{Art}\ \text{perro}_{NN}]_{FN}\ [\text{come}\ [\text{huesos}_{NN}]_{FN}]_{FV})_{O}$
- $([\text{El}_{Art}\ \text{gato}_{NN}]_{FN}\ [\text{come}_{V}\ [\text{del}_{PP}\ [\text{plato}_{NN}]_{FN}]_{FP}]_{F}V)_{O}$

Parseo sintáctico con CFG

Para realizar un parseo sintáctico (full parsing) con una CFG se proponen los siguientes pasos:

1. **Fase de aprendizaje:**
   1. Conformar el alfabeto terminal Σ y no terminal Δ a partir del corpus, así como definir símbolo inicial O.
   2. Generar las reglas, R, de la gramática a partir del etiquetado del corpus.
   3. Conformar la gramática $CFG = (\Sigma, \Delta, O, R)$, en forma normal de Chomsky.
2. **Fase de evaluación:**
   1. Tomar un conjunto de ejemplos (no necesariamente del corpus de la fase anterior) para parsear.
   2. Utilizar un algoritmo de parsing (**algoritmo de CYK**) para parsear los ejemplos.
   3. Determinar la calidad del parseo.

Algoritmo CYK

El algoritmo de CYK (Cocke-Younger-Kasami) es un algoritmo de parsing para lenguajes libres de contexto.

El algoritmo de CYK tiene las siguientes propiedades:

- Requiere de una CFG en forma normal de Chomsky.
- Crea una tabla donde, en la diagonal, asigna a cada token una etiqueta de la regla terminal $X \to w$.
- En cada parte del triángulo superior de la tabla, observa las etiquetas de la izquierda, Y_0, y de abajo, Y_1, para asignar una nueva etiquete de la regla $X \to Y_0 \cdot Y_1$.
- Su complejidad es $O(n^3|R|)$, donde n es el número de tokens de la cadena a parsear y $|R|$ el número de reglas de la CFG.

Algoritmo de CYK

Algorithm Algoritmo de CYK

```
1: procedure PARSE(string, CFG = (Σ, Δ, O, R))
2:     T ← TABLE(n × n)
3:     for j from 0 to n do
4:         for w_j in TOKENIZE(string) do:
5:             T[j, j] ← T[j, j] ∪ {X}; si X → w_j
6:         end for
7:         for i from j to 0 do
8:             for k from i to j + 1 do
9:                 T[i, j] ← T[i, j] ∪ {X}; si X → Y_0 · Y_1 and Y_0 ∈ T[i, k], Y_1 ∈ T[k + 1, j]
10:            end for
11:        end for
12:    end for
13:    return T
14: end procedure
```

Parseo sintáctico con CFG

A partir de las oraciones anteriores, podemos definir la CFG que la genera:

- Alfabeto no-terminal:
  $\Delta = \{O, FN, FP, FV, Art, PP, NN, V\}$
- Alfabeto terminal:

  $\Sigma = \{el, del, gato, perro, sopa, huesos, plato, del\}$
- Símbolo inicial: $O \in \Delta$
- Sistema de reglas R:

$$O \mapsto FN \cdot FV$$
$$FN \mapsto Art \cdot NN$$
$$FN \mapsto NN$$
$$FV \mapsto V \cdot FN | FP \cdot FP$$
$$FP \mapsto PP \cdot NN$$
$$Art \mapsto el$$
$$NN \mapsto gato | sopa | perro | hueso | plato$$
$$V \mapsto come$$
$$PP \mapsto del$$

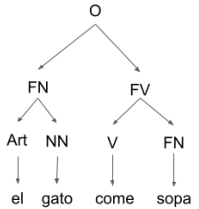
Parseo con algoritmo CYK

Consideremos la siguiente gramática en forma Normal de Chomsky:

$$
\begin{aligned}
O &\rightarrow FN \cdot FV \\
FN &\rightarrow Art \cdot NN | sopa | huesos \\
FV &\rightarrow V \cdot FN | V \cdot FP \\
FP &\rightarrow PP \cdot NN \\
Art &\rightarrow el \\
NN &\rightarrow gato | perro | plato \\
PP &\rightarrow del \\
V &\rightarrow come
\end{aligned}
$$

La oración 'el gato come sopa' da la siguiente tabla:

	el	gato	come	sopa
el	Art	FN		O
gato		NN		
come			V	FV
sopa				FN



Ambigüedad

La **ambigüedad** es un problema de los lenguajes naturales, representa una ambivalencia en su significado.

Ambigüedad

Se dice que una oración es ambigua cuando existen dos o más análisis de parsing bajo una misma gramática.

Por ejemplo, si tuvieramos reglas como $O \rightarrow FV \cdot FN | FN \cdot FV$, $FV \rightarrow FN \cdot V | V \cdot FN$, etc., habría ambigüedad en la oración:

el mazo rompió el muro

Problemas de las CFG

A pesar de que las CFG pueden manejar tareas del lenguaje natural más allá de las gramáticas regulares, sufren de algunos problemas importantes:

- Las construcciones que generan no toman en cuenta el significado de las oraciones, sólo la estructura.
- Son rígidas, no permiten desvíos de las reglas establecidas.
- No son deterministas, dada una oración, diferentes árboles pueden parsearla.

Parseo de dependencias

El parseo de dependencias etiqueta **relaciones** entre los tokens de una oración. Existen estándares de estas relaciones, algunas de ellas son:

Etiqueta	Elementos	Descripción
NSUBJ	N-V	Sujeto
DOBJ	N-V	Objeto directo
IOBJ	PP-V	Objeto indirecto
XCOMP	X-V	Complemento (tiempo, lugar, etc.)
NMOD	X-N	Modificador nominal
DET	D-N	Determinante (artículo)
CONJ	O-O	Conjunción

Parseo de dependencias basado en transiciones

El parseo de dependencias se puede realizar por medio del método **basado en dependencias**; este cuenta con los siguientes elementos:

Stack : Almacenamiento de etiquetas de las dependencia.

Buffer : Almacenamiento de los tokens de la oración que se van a parsear.

Parser : Ejecutor del parseo, el cual decide las acciones. Las acciones son:

1. LeftArc: El tope de stack es núcleo y el segundo elemento relación. remueve segundo elemento.
2. RightArc: El tope es relación y el segundo elemento núcleo. Remueve el tope.
3. Shift: Remueve el token del buffer y lo añade al stack.

Parseo de dependencias

Algorithm Algoritmo basado en transiciones

```
1: procedure DependencyParse(string)
2:     state ← [root], [Tokenize(string)], []
3:     while state not final do
4:         Action ← Parser(state)
5:         state ← Action(state)
6:     end while
7:     return T
8: end procedure
```

Parser de dependencias

Consideremos la siguiente lista de elementos:

$$Art \rightarrow el$$
$$NN \rightarrow gato|perro|plato|sopa|huesos$$
$$PP \rightarrow del$$
$$V \rightarrow come$$

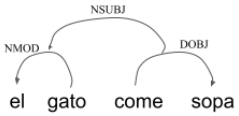
A partir de las dependencias de la gramática anterior podemos definir reglas para el parser en el parseo de dependencias:

- LeftArrow: **Si** stack[Art, NN]:DET **o** stack[NN, V]:NSUBJ
- RightArrow: **Si** stack[V, FN]:DOBJ **o** stack[V, FP]:XCOMP
- Shift: En otro caso.

Parser de dependencias

A partir de las reglas dadas, podemos parsear las dependencias en la oración "el gato come sopa":

Paso	Stack	Buffer	Parser	Resultado
0	[root]	[el, gato, come, sopa]	SHIFT	
1	[root, el]	[gato, come, sopa]	SHIFT	*el*
2	[root, el, gato]	[come, sopa]	LEFTARROW	*el* ← *gato*: DET
3	[root, gato]	[come, sopa]	SHIFT	gato
4	[root, gato, come]	[sopa]	LEFTARROW	*gato* ← *come*: NSUBJ
5	[root, come]	[sopa]	SHIFT	*come*
6	[root, come, sopa]	[]	RIGHTARROW	*come* → *sopa*: DOBJT



Shallow parsing

El shallow parsing (o chunking) consiste en obtener constituyentes a nivel superior sin llegar a una profundidad de un árbol sintáctico. Comúnmente, se obtienen chunks correspondientes a **frases**:

$$[\textit{la niña pequeña}]_{FN}\ [\textit{jugaba}]_{V}\ [\textit{con la pelota}]_{FP}$$

En particular este tipo de parseo se puede realizar con **etiquetado BIO**:

(la, B-FN) (niña, I-FN) (pequeña, I-FN) (jugaba, B-V) (con, B-FP) (la, I-FP) (pelota, I-FP)

Este etiquetado puede ser inferido por un método como **modelos ocultos de Markov** o una **red recurrente**

Gramáticas libres de contexto probabilísticas

Las gramáticas libres de contexto probabilísticas son CFG que incorporan una **distribución de probabilidad** sobre las reglas de reescritura. Se pueden considerar como **modelos gráficos**.

Gramática Libre de Contexto Probabilística

Una Gramática libre de contexto probabilística o PCFG es una cinco-tupla $PCFG = (\Sigma, \Delta, O, R, P)$, donde $O \in \Delta$ es símbolo de inicio, $\Delta = \{d_1, ..., d_n\}$ son símbolos no terminales, Σ símbolos terminales, $R = \{d_i \mapsto d_j\}$ es el conjunto de reglas y P es una medida de probabilidad, tal que si $d_j \in \Delta$ es padre de $d_i \in \Delta \cup \Sigma$ entonces $p(d_j \mapsto d_i | d_j)$.

Es claro que esta función de probabilidad cumple:

$$\sum_{i=1}^{n} (d_j \mapsto q_i | d_j) = 1$$

Probabilidad de árboles y cadenas

La probabilidad de un árbol $T(w)$ generado por la PCFG es:

$$p(T(w)|PCFG) = p(O)\prod_{i=1}^{n} p(d_j \mapsto d_i|d_j)$$

Una cadena $w \in \Sigma^+$ puede ser generado por diferentes árboles. La probabilidad de la cadena es, entonces:

$$P(w) = \sum_{T(w)} p(T(w)|PCFG)$$

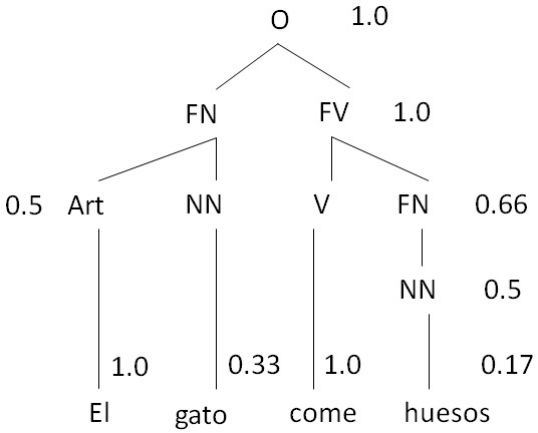
Ejemplo: PCFG

Considérese el siguiente sistema de reglas con probabilidades asignadas:

$O \mapsto FN \cdot FV$	1.0		
$FN \mapsto Art \cdot NN$	0.5	$NN \mapsto gato$	0.33
$FN \mapsto NN$	0.5	$NN \mapsto sopa$	0.17
$FV \mapsto V \cdot FN$	0.66	$NN \mapsto perro$	0.17
$FV \mapsto FP \cdot FP$	0.33	$NN \mapsto huesos$	0.17
$FP \mapsto PP \cdot NN$	1.0	$NN \mapsto plato$	0.17
$Art \mapsto el$	1.0	$V \mapsto come$	1.0
$NN \mapsto gato$	0.33	$PP \mapsto del$	1.0

Ejemplo: PCFG

El árbol generado a partir de parsear la oración "'El gato come huesos"' es:



Textos recomendados

Clark, S., Fox, C. y Lappin, S. (2013). "1. Formal Language theory". *The Handbook of Computational Linguistics and Natural Language Processing*. John Wiley and Sons, pp. 11-42.

Abney, S. (1991). 'Parsing by chunks'. *Principle-based parsing*. Springer, pp. 257-278.

Hopcroft, J., Motwani, R. y Ullman, J. (2001). *Introduction to automata theory, languages, and computation*. Pearson.

Jurafski, D. y Martin, J. (2018). "2. Regular Expression, Text Normalization, Edit Distance". *Speech and Language Processing*. Pearson.

Kracht, M. (2011). *The mathematics of language*. Cambridge University Press.