

# Gramáticas formales

Formalismos e implementaciones

Fernando Carranza

Pablo Zdrojewski

Macarena Fernández Urquiza

Julia Milanese

Draft  
of April 24, 2023, 11:36

No series description provided

# Gramáticas formales

Formalismos e implementaciones

Fernando Carranza

Pablo Zdrojewski

Macarena Fernández Urquiza

Julia Milanese

Fernando Carranza

bibnamedelimb Pablo Zdrojewski

bibnamedelimb Macarena Fernández Urquiza Julia Milanese. 2023. *Gramáticas formales: Formalismos e implementaciones*. Berlin: Language Science Press.

This title can be downloaded at:

redefine\lsURL

© 2023, Fernando Carranza

bibnamedelimb Pablo Zdrojewski

bibnamedelimb Macarena Fernández Urquiza Julia Milanese

Published under the Creative Commons Attribution 4.0 Licence (CC BY 4.0):

<http://creativecommons.org/licenses/by/4.0/>

ISBN: no digital ISBN

no print ISBNs!

no DOI

ID not assigned!

Cover and concept of design: Ulrike Harbort

Fonts: Libertinus, Arimo, DejaVu Sans Mono

Typesetting software: Xe<sub>La</sub>TeX

redefine \publisherstreetaddress

redefine \publisherurl

Storage and cataloguing done by redefine \storageinstitution

# Contents

<b>I</b>	<b>Nociones básicas de teoría de los lenguajes</b>	<b>1</b>
<b>1</b>	<b>Sobre la formalización</b>	<b>3</b>
1.1	Introducción . . . . .	3
1.2	Conjuntos . . . . .	5
1.2.1	Definición de conjuntos . . . . .	5
1.2.2	Relaciones entre conjuntos . . . . .	8
1.2.3	Operaciones entre conjuntos . . . . .	10
1.2.4	Diagramas de Venn . . . . .	11
1.3	Funciones . . . . .	13
1.3.1	Relaciones . . . . .	13
1.3.2	Definición de función . . . . .	14
1.3.3	Clasificación de funciones . . . . .	15
1.4	Equivalencia entre funciones y conjuntos . . . . .	16
1.5	Ejercicios . . . . .	16
1.6	Recapitulación . . . . .	17
1.7	Lecturas recomendadas . . . . .	18
<b>2</b>	<b>Los lenguajes formales</b>	<b>19</b>
2.1	Introducción . . . . .	19
2.2	Alfabetos, cadenas y lenguajes . . . . .	20
2.3	Operaciones . . . . .	23
2.3.1	Operaciones entre cadenas . . . . .	23
2.3.2	Operaciones entre lenguajes . . . . .	24
2.4	Lenguajes como problemas . . . . .	24
2.5	Sobre la relevancia lingüística de los lenguajes . . . . .	25
2.6	Ejercicios . . . . .	25
2.7	Recapitulación . . . . .	29
2.8	Lecturas recomendadas . . . . .	29
<b>3</b>	<b>Jerarquía de los lenguajes formales</b>	<b>31</b>
3.1	Introducción . . . . .	31

## Contents

3.2	Jerarquía de lenguajes formales . . . . .	32
3.3	Formas de definir lenguajes . . . . .	33
3.3.1	Autómatas . . . . .	34
3.3.2	Las gramáticas . . . . .	36
3.4	Clases de lenguajes . . . . .	38
3.4.1	Clases de lenguajes por su capacidad discriminadora . .	38
3.4.2	Clases de lenguajes por su capacidad de determinar pertenencia . . . . .	45
3.4.3	Algunos lenguajes populares . . . . .	46
3.4.4	Nociones básicas de complejidad . . . . .	46
3.5	Ejercicios . . . . .	48
3.5.1	Autómatas . . . . .	48
3.5.2	Expresiones regulares . . . . .	49
3.5.3	Gramáticas regulares . . . . .	50
3.5.4	Tipos de reglas gramaticales . . . . .	51
3.5.5	Repaso . . . . .	51
3.6	Recapitulación . . . . .	53
3.7	Lecturas recomendadas . . . . .	53
<b>II</b>	<b>Las gramáticas basadas en constituyentes</b>	<b>55</b>
<b>4</b>	<b>Gramáticas independientes de contexto</b>	<b>57</b>
4.1	Introducción . . . . .	57
4.2	Gramáticas independientes de contexto . . . . .	58
4.2.1	Definición . . . . .	58
4.2.2	Derivación . . . . .	59
4.2.3	Árboles . . . . .	60
4.2.4	Axiomas de dominancia y precedencia . . . . .	61
4.2.5	Chomsky Normal Form . . . . .	66
4.2.6	Algunas limitaciones de las CFG . . . . .	66
4.2.7	Ejercitación . . . . .	69
4.2.8	Parsers . . . . .	69
4.2.9	Parsers sintáctico . . . . .	70
4.3	Recapitulación . . . . .	71
4.4	Lecturas recomendadas . . . . .	71

<b>5</b>	<b>Gramática independiente de contexto generalizada (GPSG)</b>	<b>73</b>
5.1	Introducción . . . . .	73
5.1.1	Convenciones de las Gramáticas independientes de contexto . . . . .	74
5.1.2	Problemas conceptuales de las Gramáticas independientes de contexto . . . . .	75
5.2	GPSG: el formalismo . . . . .	76
5.2.1	Las reglas ID/LP . . . . .	77
5.2.2	El uso de rasgos y sus convenciones asociadas . . . . .	78
5.2.3	Las metarreglas . . . . .	80
5.2.4	Reglas léxicas . . . . .	81
5.2.5	La semántica . . . . .	82
5.2.6	Los postulados de significado . . . . .	84
5.2.7	Las dependencias no acotadas . . . . .	85
5.2.8	Sobre el poder expresivo de GPSG . . . . .	87
5.3	Ejercicios . . . . .	88
5.3.1	Reglas para gramática independientes de contexto . . . . .	88
5.3.2	Gramáticas independientes de contexto y linealidad . . . . .	88
5.3.3	<i>Local reordering</i> en alemán . . . . .	89
5.3.4	Un fragmento para GPSG (rasgo SUBCAT) . . . . .	90
5.3.5	Metarregla y reglas ID . . . . .	92
5.3.6	Aplicación funcional . . . . .	93
5.4	Recapitulación . . . . .	94
5.5	Recomendación bibliográfica . . . . .	94
<b>III</b>	<b>Gramáticas basadas en constituyentes</b>	<b>95</b>
<b>6</b>	<b>Gramáticas minimalistas</b>	<b>97</b>
6.1	Introducción . . . . .	97
6.2	Breve reseña histórica . . . . .	98
6.2.1	Rección y Ligamiento . . . . .	98
6.3	Programa Minimalista . . . . .	99
6.3.1	Algunas definiciones previas . . . . .	99
6.3.2	Minimalist Grammar . . . . .	101
6.4	Ejercicios . . . . .	108
6.4.1	Ejercicio . . . . .	108
6.4.2	Ejercicio . . . . .	108
6.4.3	Ejercicio . . . . .	109

*Contents*

6.4.4	Ejercicio . . . . .	109
6.4.5	Ejercicio . . . . .	110
6.4.6	Ejercicio . . . . .	110
<b>References</b>		<b>111</b>
<b>Index</b>		<b>115</b>
	Name index . . . . .	115



Lo que sigue es un primer borrador de un manual de formalismos gramaticales. Este manual surgió inicialmente como notas de clase del seminario “Gramáticas formales: formalismos e implementaciones”, dictado durante el primer cuatrimestre del 2022 en el marco de la carrera de Letras de la Facultad de Filosofía y Letras de la Universidad de Buenos Aires.



## **Part I**

# **Nociones básicas de teoría de los lenguajes**



# 1 Sobre la formalización

## 1.1 Introducción

Este libro, como se sigue del título, es un libro sobre gramáticas formales. Cabe entonces empezar preguntándose precisamente qué son las gramáticas formales. Una forma de abordar esta pregunta es primeramente clarificar qué se entiende por gramática, para, en segundo lugar, ver cuáles son las características que permiten que una pueda describirse como formal.

En el lenguaje especializado podemos identificar al menos cuatro acepciones diferentes para la palabra “gramática”. La primera es la de gramática como disciplina científica, es decir, como un campo de investigación dentro de la lingüística. Para algunas escuelas, se trata de la más importante de sus ramas, puesto que es la responsable de explicar lo que Humboldt denominaba “la forma del lenguaje”, es decir, el sistema de leyes básicas que subyace a la generación lingüística. La segunda acepción es la de gramática como un sistema de conocimiento de naturaleza mental, responsable de que el ser humano sea capaz de producir e interpretar las oraciones de su lengua. La tercera acepción es la de gramática entendida como un modelo lógico matemático que pretende dar cuenta de cómo es posible la generación e interpretación lingüística. La cuarta es la de la gramática entendida como modelo teórico. Así, son gramáticas en este sentido la gramática generativa (Chomsky 1957 y trabajos posteriores), Head-driven Phrase Structure Grammar (HPSG, de ahora en más, ver Pollard & Sag 1994), Lexical Funcional Grammar (LFG de ahora en más, Bresnan 1982), la gramática cognitiva (Langacker 1987, 1991), entre muchas otras.

El término “formal” refiere, por su parte, a la formalización. Una gramática formal se define, principalmente, como un modelo que recurre a algún formalismo, es decir, a algún tipo de notación lógico matemática, y que se caracteriza por la explicitud, es decir, por la capacidad que tiene de permitirnos demostrar como teoremas las oraciones que genera.

Existe una tradición académica que opone los enfoques formales a los enfoques funcionales. Creemos que tal oposición es incorrecta. Siguiendo observaciones de Partee (2016: 3) y Bach (1989: 9) con respecto a la semántica formal, podemos decir que la gramática formal no se opone a la gramática funcional,

sino a lo que podríamos denominar la “gramática informal”. Una gramática informal sería lo que Di Tullio (1990: 4) denomina una gramática preteórica, es decir, un mero conjunto de observaciones. Las gramáticas formales, en cambio, se caracterizan por volcar esas observaciones en un sistema axiomático. Siguiendo a Shieber (1986: 38), podemos establecer una distinción entre formalismos que tienen pretensiones teóricas, es decir, en los que hay una asociación entre el modelo lógico matemático y un modelo teórico, o formalismos que son meramente un sistema de notación útil para algún fin, por ejemplo, para su implementación computacional. En el primer caso hablamos de gramáticas formales teóricas y, en el segundo, de gramáticas formales prácticas<sup>1</sup>.

Si bien resulta obvio por qué una gramática formal práctica recurriría a la formalización, sí cabría preguntarse qué sentido tiene para las gramáticas formales teóricas recurrir a ella. Para responder a esta pregunta, Peregrín Otero (1970: 77) recurre al viejo acertijo del gavián y las cien palomas. Supongamos que un gavián sentado en una rama ve pasar una bandada de palomas. Al verlas, este dice “Adiós, mis cien palomas”. Pero una de ellas le responde lo siguiente:

- (1) No somos cien. Si sumamos las que somos, más tantas como las que somos, más la mitad de las que somos, y la mitad de la mitad de las que somos, en ese caso, contigo, gavián, seríamos cien.

El acertijo es entonces preguntarse cuántas palomas había en la bandada. A primera vista, puede parecer un problema difícil de resolver. Sin embargo, esta enorme dificultad se empieza a desvanecer en la medida en que vemos que la respuesta de la paloma se puede representar como una suma en la que cada una de las partes del problema puede traducirse en los siguientes términos:

- (2) a. las que somos =  $n$   
b. tantas como las que somos =  $n$   
c. la mitad de las que somos =  $\frac{n}{2}$   
d. la mitad de la mitad de las que somos =  $\frac{n}{4}$   
e. contigo, gavián = 1

Podemos coleccionar todas estas partes y conformar la siguiente fórmula:

$$(3) \quad n + n + \frac{n}{2} + \frac{n}{4} + 1 = 100$$

---

<sup>1</sup>Shieber 1986 las describe como “gramáticas como herramientas”. Para mantener el paralelismo, nos resulta más útil denominarlas gramáticas formales prácticas.

Y ahora solo es cuestión de despejar la  $n$  para dar con el resultado final, que es 36. Como vemos en este ejemplo, la formalización es un modo de representar un problema de modo de entenderlo mejor y poder solucionarlo. En esta misma línea, las gramáticas formales teóricas son escuelas gramaticales que confían en el papel de la formalización como medio para comprender cabalmente su objeto de estudio y cumplir con lo que espera de una teoría científica, que es hacer afirmaciones que generen predicciones que puedan ser contrastadas con los datos.

En suma, formalizar es representar determinado fenómeno en términos de un sistema riguroso y explícito. Para formalizar una lengua, las gramáticas formales, tanto teóricas como prácticas, utilizan un sistema axiomático que recurre para su expresión a herramientas de la lógica y de la matemática. Se espera que las oraciones de la lengua en cuestión puedan derivarse como teoremas a partir de dicho sistema. En lo que sigue de este capítulo, el principal objetivo es introducir las nociones de la lógica y de la matemática más importantes para poder llevar adelante esa tarea, y que se retomarán a lo largo de todo el libro. Para eso, en la sección 1.2 presentamos los rudimentos básicos de la teoría de conjuntos y, en la sección 1.3, el concepto de función.

## 1.2 Conjuntos

Un conjunto es, sencillamente, una colección de objetos. Cada uno de estos objetos son los *miembros* o *elementos* de ese conjunto. En un conjunto no es posible repetir objetos, cada miembro es necesariamente único. Por fuera de esa condición, no hay mayores restricciones para la membresía. Un conjunto puede estar conformado por elementos tan disímiles como autos, perros, letras, números naturales, oraciones del español o monstruos mitológicos. Lo importante, para poder reconocer a un elemento como miembro de un conjunto particular, es que se lo haya definido como tal. Además, así como no hay elementos repetidos, en un conjunto tampoco hay un orden de sus elementos. Es indistinto mencionar a un miembro antes que otro o hacerlo de forma inversa.

### 1.2.1 Definición de conjuntos

Los conjuntos pueden definirse por *extensión*, por *intensión* o, si se trata de conjuntos recursivos, por *reglas recursivas*. La notación por extensión consiste en listar todos los elementos que pertenecen al conjunto. Por ejemplo, de la siguiente forma puedo definir el conjunto de los útiles escolares:

- (4) {lapicera, lápiz, regla, cuaderno, goma}

## 1 Sobre la formalización

Recordemos que cualquier elemento puede pertenecer a un conjunto y que sus miembros no necesariamente deben tener algo en común, por lo que el siguiente también es un conjunto válido:

- (5) {Facultad de Filosofía y Letras, 10, Python}

Algo importante a tener en cuenta cuando definimos un conjunto con los nombres de los elementos que lo conforman es que *lo que pertenece al conjunto es el elemento denotado por el nombre, no el nombre en sí mismo*. En el ejemplo anterior, lo que pertenece al conjunto es la Facultad de Filosofía y Letras y no su nombre. Lo mismo con el lenguaje de programación Python o con el número 10.

Si, en cambio, lo que quisiésemos incluir en el conjunto fuese el nombre de determinado elemento, lo que debemos hacer es utilizar comillas simples:

- (6) {'Facultad de Filosofía y Letras', 10, Python}

Ahora nuestro conjunto es el conjunto del nombre de la Facultad de Filosofía y Letras, el número 10 y el lenguaje de programación Python.

Esto nos abre la posibilidad de definir un conjunto por extensión haciendo uso de distintos nombres o formas de designar y que, sin embargo, todas esas formas aludan al mismo conjunto:

- (7) a. {Ciudad Autónoma de Buenos Aires, La Plata, Santa Fe}  
b. {Capital Federal, La Plata, Santa Fe}  
c. {La capital de la República Argentina, La Plata, Santa Fe}

Por último, cabe hacer dos aclaraciones. En primer lugar, dado que los conjuntos no tienen orden, es indistinto el orden en que se nombre sus miembros. De esta manera, (8a) y (8b) son exactamente el mismo conjunto.

- (8) a. {Tucumán, Salta, Jujuy}  
b. {Salta, Jujuy, Tucumán}

En segundo lugar, puesto que cada miembro es único, es redundante repetir un elemento. De esta manera, los conjuntos en (9) son idénticos:

- (9) a. {Pasco, San Lorenzo, Tucumán, Huaqui}  
b. {Pasco, San Lorenzo, Pasco, Tucumán, Huaqui, Huaqui}  
c. {Pasco, Pasco, Pasco, Pasco, San Lorenzo, San Lorenzo, Tucumán, Tucumán, Huaqui}



d. {Pasco, San Lorenzo, San Lorenzo, Tucumán, Huaqui, Huaqui, Huqui}

Ahora bien, imaginemos que nuestro conjunto es el conjunto de los números múltiplos de 2 mayores a 2 y menores a 50. En este caso nos sería un poco más costoso nombrarlos a todos. Podríamos recurrir a cierta abreviatura:

(10) {4, 6, 8, 10, ..., 50}

Pero en este caso sería más útil recurrir a la *notación por predicados* (también llamada *por comprensión* o *por intensión*), que establece una propiedad que los miembros de un conjunto deben tener de modo que puedan ser reconocidos como tales. Siguiendo nuestro ejemplo:

(11) { $x \mid x$  es múltiplo de 2, mayor a 2 y menor o igual a 50}

La línea vertical o *pleca* se lee “tal que” y marca el inicio de la definición de la propiedad. La “ $x$ ” no refiere a ningún elemento en particular. Es una variable cuyo valor es asignado de acuerdo a la propiedad indicada a continuación. En esta notación también es posible utilizar dos puntos (:) en lugar de la pleca:

(12) { $x$ :  $x$  es múltiplo de 2, mayor a 2 y menor o igual a 50}

Por último, si un conjunto es recursivo<sup>2</sup>, este puede definirse mediante *reglas recursivas*. Por ejemplo, el conjunto  $E$  de los números múltiplos de 2 mayores a 2 puede definirse como sigue:

- (13)
- a.  $4 \in E$
  - b. Si  $x \in E$ , entonces  $x+2 \in E$
  - c. Nada más pertenece a  $E$

El símbolo  $\in$  debe leerse como “pertenece” e indica que un elemento es miembro de un conjunto. La primera regla en (13) nos dice que 4 pertenece al conjunto  $E$ . La segunda regla, luego, nos indica que, si un número pertenece al conjunto, ese número incrementado en 2 también es miembro de tal conjunto. Finalmente, la tercera regla indica que ningún otro elemento es miembro del conjunto en cuestión. Esta última regla se conoce típicamente como *Regla de clausura* y hace que un conjunto sea recursivo en tanto delimita claramente que los elementos anteriormente indicados pertenecen al conjunto y el resto no pertenecen. Esta

---

<sup>2</sup>La noción de recursividad se abordará con más detalle en el capítulo 3. Por ahora, basta tener presente que un conjunto es recursivo si, dado cierto elemento, es posible identificar si pertenece o no al conjunto en cuestión.


regla es muy importante, puesto que si faltara, solo podríamos tener certeza de los miembros del conjunto y no de los elementos que no le pertenecen.

Dadas estas tres notaciones, es posible usar cualquiera de ellas para definir un conjunto. Por supuesto, algunas notaciones pueden ser más convenientes que otras. Si el conjunto que queremos definir tiene infinitos elementos, la notación extensional no será la óptima, dado que no seremos capaces de nombrar a todos sus miembros. Por otro lado, si el conjunto es finito y sus elementos no tienen ninguna cualidad en común más que ser miembros de tal conjunto, posiblemente esta notación sí sea la más útil.

Cuando dos conjuntos, definidos con cualquier notación, tienen exactamente los mismo elementos, decimos que se trata del *mismo conjunto*. Eso puede expresarse con el signo '='.

$$(14) \quad \{1, 2, 3\} = \{x \mid x \in \mathbb{N}^3 \text{ y } 0 < x < 4\}$$

De esto podemos deducir que el conjunto vacío es único. Existe solamente un conjunto que no posee ningún elemento. Para indicar que nos referimos a él utilizamos el símbolo  $\emptyset$ .

 Ver ejercicio en [1.1](#).

### 1.2.2 Relaciones entre conjuntos

Cuando los miembros de un conjunto  $A$  también son miembros de un conjunto  $B$ , decimos que  $A$  es un *subconjunto* de  $B$  o, también que  $A$  *está incluido en*  $B$ . A esta relación la denotamos de la siguiente manera:  $A \subseteq B$ . El subconjunto  $B$ , en este caso, podría contener elementos que  $A$  no, pero esto no es obligatorio, por lo que la *relación de subconjunto* permite que cada conjunto sea un subconjunto de sí mismo (i.e.  $A \subseteq A$ ).

Si, en cambio, lo que deseamos es indicar que  $A$  es subconjunto de  $B$  y que este último tiene efectivamente elementos que el primero no (i.e. son conjuntos distintos), debemos utilizar la *relación de subconjunto propio*, la cual se denota:  $A \subset B$ .

Algunos ejemplos de estas relaciones son:

- (15)    a.  $\{a, b, c\} \subseteq \{a, b, c, g, h, z\}$   
          b.  $\{a, b, c\} \subseteq \{a, b, c\}$   
          c.  $\{a, b\} \not\subseteq \{h, i\}$

---

<sup>3</sup>El símbolo  $\mathbb{N}$  se usa para denotar el conjunto de los números naturales.

- d.  $\{m, n\} \subset \{m, n, o\}$
- e.  $\emptyset \subset \{p\}$
- f.  $\{\emptyset\} \not\subset \{p\}$
- g.  $\{\{a\}\} \not\subset \{a\}$
- h.  $\{a\} \not\subset \{\{a\}\}$ , pero  $\{a\} \in \{\{a\}\}$

De los anteriores ejemplos, detengámonos en los últimos 4. La definición de subconjunto nos permite afirmar que el conjunto vacío está incluido (de forma propia o impropia) en cualquier otro conjunto (ejemplo (15e)), dado que un conjunto es subconjunto de otro si todos sus miembros se encuentran contenidos en este último. Puesto que el conjunto vacío no posee miembro alguno, esto es trivialmente cierto.

No sucede lo mismo, en cambio, con el conjunto que contiene al conjunto vacío (ejemplo (15f)). Si un conjunto  $A$  contiene al conjunto vacío, entonces tiene un miembro, y si este elemento no se encuentra en un conjunto  $B$ , entonces *no puede decirse que  $A$  esté incluido en  $B$* .

Algo similar ocurre con un conjunto que contiene a otro conjunto, llamémosle  $A$ . Este nunca será subconjunto de otro conjunto  $B$  si este último no posee también a dicho conjunto en cuestión.

Por último, el ejemplo (15h) nos permite reflexionar sobre la diferencia entre un subconjunto y un miembro de un conjunto. *El subconjunto es siempre un conjunto, mientras que un miembro de un conjunto puede o no ser un conjunto en sí mismo*. Por ejemplo,  $a$  es un miembro del conjunto  $\{a, b, c\}$ , pero no es un subconjunto de este dado que no es un conjunto en absoluto. El conjunto  $\{a\}$ , en cambio, sí lo es y es también un subconjunto (propio, de hecho) de  $\{a, b\}$ . Vemos así que la noción de inclusión afecta a los conjuntos mientras que la de pertenencia, a los elementos o miembros.

Respecto de esto, vale la pena detenernos en una última diferencia respecto de estas relaciones: la propiedad de *transitividad*. Mientras que la inclusión es transitiva por definición, la pertenencia no lo es. Por ejemplo, si  $X$  y  $C$  son dos subconjuntos tales que  $X = \{a, b, c\}$  y  $C = \{a, X\}$ , podemos afirmar que  $b \in X$  y  $X \in C$  pero  $b \notin C$ , dado que  $C = \{a, \{a, b, c\}\}$ . En cambio, si consideramos los conjuntos  $A$ ,  $B$  y  $C$  tales que  $A \subseteq B$  y  $B \subseteq C$ , por la misma definición de subconjunto  $A \subseteq C$ . Esto se debe a que, si  $A$  está incluido en  $B$ , todos los elementos de  $A$  están en  $B$  y, si  $B$  está incluido en  $C$ , todos los elementos de  $B$  están también en  $C$ , por lo que todos los elementos de  $A$  estarán necesariamente en  $C$ .

### 1.2.3 Operaciones entre conjuntos

Las siguientes operaciones toman dos conjuntos y devuelven otro conjunto. En primer lugar, la *unión* es aquella operación que toma a dos conjuntos y devuelve como resultado un conjunto compuesto por los elementos de los primeros dos. Se simboliza mediante el operador  $\cup$ :

$$(16) \quad A \cup B = \{x | x \in A \text{ o } x \in B\}$$

Por ejemplo: Sea  $A=\{a,b,c\}$  y  $B=\{c,h,i\}$ ,  $A \cup B=\{a,b,c,h,i\}$ .

En segundo lugar, la *intersección* toma dos conjuntos y devuelve un conjunto conformado por los elementos presentes en ambos. Se simboliza mediante el operador  $\cap$ :

$$(17) \quad A \cap B = \{x | x \in A \text{ y } x \in B\}$$

Por ejemplo: Sea  $A=\{a,b,c\}$  y  $B=\{c,h,i\}$ ,  $A \cap B=\{c\}$ .

En tercer lugar, la *diferencia*, es una operación que subtrae los elementos de un conjunto a otro. Se simboliza mediante el operador  $-$ .

$$(18) \quad A - B = \{x | x \in A \text{ y } x \notin B\}$$

Por ejemplo: Sea  $A=\{a,b,c\}$  y  $B=\{c,h,i\}$ ,  $A - B=\{a,b\}$ .

Es importante aquí notar que, a diferencia de lo que sucede con las dos operaciones anteriores, el orden en el que se aplica la substracción es relevante. Sin importar cuáles sean los conjuntos  $A$  y  $B$ , vale que  $A \cup B = B \cup A$  y  $A \cap B = B \cap A$ . Sin embargo,  $A - B \neq B - A$ .

En cuarto lugar, la operación *complemento* toma un conjunto y devuelve todos los elementos que no pertenezcan a él. Existen diferentes formas de simbolizar esta operación. Acá vamos a optar por una barra horizontal sobre el conjunto al cual se aplica.

$$(19) \quad \bar{A} = \{x | x \notin A\}$$

Los elementos que no pertenezcan a  $A$  pero sí a su complemento dependerán del universo o dominio con el que se esté trabajando. Por ejemplo, si el universo son los números naturales y  $A=\{x \mid x \text{ es un número natural par}\}$ ,  $\bar{A}$  estará conformado por todos los números naturales impares.

En quinto lugar, el conjunto potencia de  $A$  es el conjunto de todos los conjuntos posibles que se pueden construir en función de los miembros de  $A$ . Por ejemplo,

si  $A = \{a, b, c\}$ , el conjunto potencia de  $A$ ,  $P(A)$ , equivale al conjunto  $\{\{a,b,c\}, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}\}$ .

Por último, el *producto cartesiano* es aquella operación que toma dos conjuntos  $A$  y  $B$  y genera todos los pares ordenados que toman como primer elemento un miembro del primer conjunto y, como segundo elemento, un miembro del segundo. Esta operación se escribe  $A \times B$  y se define del siguiente modo:

$$(20) \quad A \times B = \{\langle x, y \rangle \mid x \in A \text{ e } y \in B\}$$

Por ejemplo, si  $A = \{m, n, o\}$  y  $B = \{4, 5\}$ , entonces:

$$(21) \quad \begin{aligned} \text{a. } A \times B &= \{\langle m, 4 \rangle, \langle m, 5 \rangle, \langle n, 4 \rangle, \langle n, 5 \rangle, \langle o, 4 \rangle, \langle o, 5 \rangle\} \\ \text{b. } B \times A &= \{\langle 4, m \rangle, \langle 4, n \rangle, \langle 4, o \rangle, \langle 5, m \rangle, \langle 5, n \rangle, \langle 5, o \rangle\} \\ \text{c. } B \times B &= \{\langle 4, 4 \rangle, \langle 4, 5 \rangle, \langle 5, 4 \rangle, \langle 5, 5 \rangle\} \end{aligned}$$

A diferencia de un conjunto, que como ya se señaló no tiene orden, un par ordenado sí lo tiene, por lo que si bien el producto cartesiano, al ser un conjunto, no tiene orden, sus elementos sí tienen un orden interno.

Cabe también aclarar que el producto cartesiano de un conjunto cualquiera con el conjunto vacío  $\emptyset$  da como resultado el conjunto vacío.

#### 1.2.4 Diagramas de Venn

Los diagramas de Venn son una buena forma de representar conjuntos. En estos diagramas, cada conjunto se representa con un círculo y sus miembros, con puntos. En la figura 1.1, la región 1 contiene los elementos que pertenecen al conjunto  $A$  pero no al  $B$ , y lo contrario sucede con la región 2. La tercera región indica los elementos que pertenecen tanto al conjunto  $A$  como al  $B$  (i.e. la intersección de ambos). Y la región 4 representa los elementos que no se encuentran en ninguno de los conjuntos.

Aquí recurrimos la denominación de estas operaciones mediante números a los fines prácticos de la explicación. Pero cuando queremos indicar que nos referimos a determinados elementos en particular es más usual utilizar el sombreado de la región en cuestión. En la figura 1.2, ejemplificamos con las distintas operaciones.

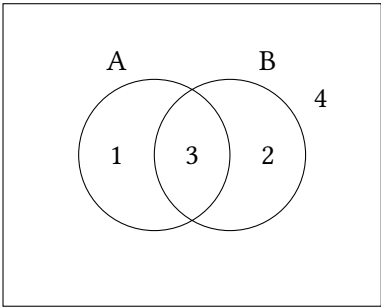


Figure 1.1: Diagrama de Venn

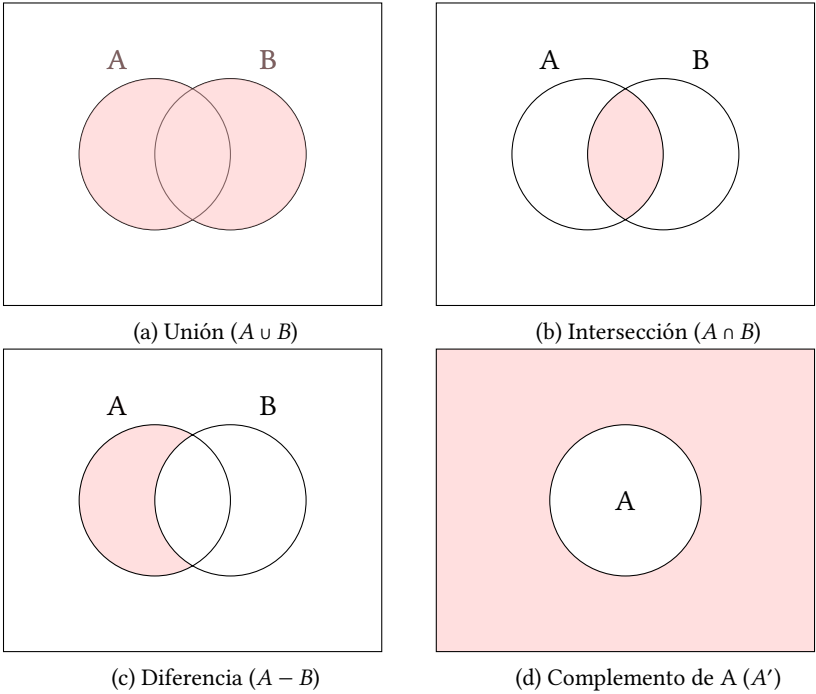


Figure 1.2: Diagramas de Venn: ejemplos de operaciones

## 1.3 Funciones

### 1.3.1 Relaciones

Intuitivamente, podríamos decir que una relación es algún tipo de vínculo que dos elementos tienen o no<sup>4</sup>. La maternidad, por ejemplo, es una relación que mantiene una madre con sus hijos o hijas, pero que no mantienen estos últimos entre sí. Del mismo modo, un conjunto puede ser subconjunto de otro. Pero un elemento que no sea un conjunto nunca podrá ser un subconjunto de otra colección. Así, los miembros de un conjunto pueden tener relaciones con elementos del mismo o de otro conjunto. Para la noción de relación, usamos la siguiente notación:  $Rab$  o  $aRb$ , donde  $R$  es la relación en cuestión, y  $a$  y  $b$  son los elementos involucrados en ella. También es posible anotar  $R \subseteq A \times B$  para indicar que el primer elemento de la relación pertenece al conjunto  $A$  y el segundo, a  $B$ . En este caso, se dice que la relación es *de*  $A$  *a*  $B$ . La figura 1.3 representa visualmente esta situación. Allí, las flechas indican las relaciones entre los distintos elementos. Cuando la relación ocurre entre elementos que pertenecen al mismo conjunto, supongamos  $A$ , decimos que es una relación *en*  $A$ .

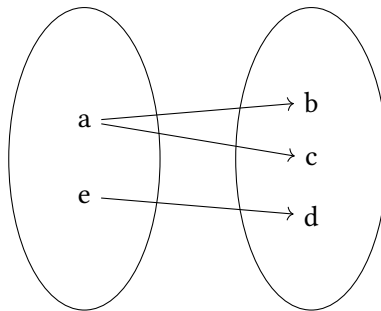


Figure 1.3: Relación  $R : A \rightarrow B$

La figura 1.3 nos muestra dos conjuntos y relaciones que se establecen entre algunos de sus elementos. En concreto,  $\{ \langle a, b \rangle, \langle a, c \rangle, \langle e, d \rangle \}$ . El *complemento de una relación*  $R \subseteq A \times B$ , denotado  $R'$ , está dado por los pares ordenados resultantes del producto cartesiano *no* contemplados por esa relación:

$$(22) \quad \bar{R} = (A \times B) - R$$

<sup>4</sup>En realidad, las relaciones pueden darse entre más de dos elementos también. Sin embargo, en esta presentación nos limitaremos a las relaciones binarias.

## 1 Sobre la formalización

En nuestro ejemplo,  $\bar{R} = \{\langle a, d \rangle, \langle e, b \rangle, \langle e, c \rangle\}$ .

Por otro lado, la **inversa de una relación**  $R \subseteq A \times B$ , indicada con  $R^{-1}$ , contiene los pares ordenados que pertenecen a  $R$  pero con sus coordenadas invertidas. Siguiendo el ejemplo,  $R^{-1} = \{\langle b, a \rangle, \langle c, a \rangle, \langle d, e \rangle\}$ .

Observemos que  $\bar{\bar{R}}$  es idéntico a  $R$  y que  $(R^{-1})^{-1} = R$ . Pero, mientras que  $R \subseteq A \times B$  y  $\bar{R} \subseteq A \times B$ ,  $R^{-1} \subseteq B \times A$ .

Conviene tener presente la siguiente nomenclatura. El *dominio* de una función es el conjunto que contiene los elementos “de los que parte” una relación. El *rango* o *imagen* es el conjunto que contiene los elementos “a los que efectivamente llega” una relación. En tercer lugar, el *codominio* es el conjunto que contiene al rango. Codominio y rango pueden ser el mismo conjunto o no<sup>5</sup>.

### 1.3.2 Definición de función

Una función es un tipo especial de relación. Se dice que una relación entre  $A$  y  $B$  es una función si y solo si se cumplen las siguientes dos propiedades: (i) Cada elemento en el dominio está emparejado solamente con un elemento en el rango, (ii) el dominio de  $R$  es igual a  $A$ .

Lo anterior es equivalente a decir que el producto cartesiano entre  $A$  y  $B$  ( $A \times B$ ) es una función solamente si todos los miembros de  $A$  aparecen una única vez en la primera coordenada de un par ordenado.

Si suponemos que  $A = \{m, n, o\}$  y  $B = \{1, 2, 3, 4, 5\}$ , los siguientes son ejemplos de funciones de  $A$  a  $B$ :

- $M = \{\langle m, 1 \rangle, \langle n, 3 \rangle, \langle o, 5 \rangle\}$
- $N = \{\langle m, 1 \rangle, \langle n, 2 \rangle, \langle o, 3 \rangle\}$

Obsérvese que en ninguno de los dos casos los elementos de  $B$  aparecen en todas las tuplas, pero eso no es un problema. Lo importante es que todos los elementos de  $A$  estén en una y no más de una.

Los siguientes ejemplos, en cambio, no constituyen funciones:

- (23) a.  $P = \{\langle m, 1 \rangle, \langle n, 2 \rangle\}$   
b.  $Q = \{\langle m, 1 \rangle, \langle n, 2 \rangle, \langle o, 3 \rangle, \langle m, 4 \rangle\}$

---

<sup>5</sup>Por ejemplo, una relación que toma cualquier valor de los números naturales ( $\mathbb{N}$ ) y lo multiplica por 2 devuelve también un valor contenido en los números naturales ( $\mathbb{N}$ ), este será su codominio. Sin embargo, su rango estará formado por los números naturales pares exclusivamente.



En el primer caso, no todos los miembros de  $A$  integran un par ordenado y en el segundo, el elemento  $m$  se encuentra asociado a dos elementos en el rango.

La terminología para hablar de funciones es muy similar a la que usamos para anteriormente para hablar de relaciones. Cuando una función toma como valores de entrada elementos del conjunto  $A$  y como resultado devuelve elementos del conjunto  $B$ , decimos que esa función *va de  $A$  a  $B$*  o, lo que es lo mismo, notamos  $f : A \rightarrow B$ .

Los elementos dentro del dominio de la función se suelen llamar **argumentos** y los que se encuentran entre su rango, **valores**. De este modo, en nuestro ejemplo anterior  $M(m) = 1$ ,  $m$  consituye el argumento y 1, el valor resultante de aplicar la función  $M$  a  $m$ .

Por último, no debemos perder de vista que, si bien aquí hemos utilizado ejemplos de funciones unarias, esto ha sido solo a los efectos de simplificar la explicación. Una función puede tomar más de un argumento. Pensemos, por ejmplo, en la función suma:  $f(x, y) = x + y$ .

### 1.3.3 Clasificación de funciones

Supongamos  $f : A \rightarrow B$ . Según su comportamiento, esta función podrá clasificarse de la siguiente manera:

- Si codominio y rango coinciden, se dice que la función es **sobreyectiva** (*A onto B*). Si esto no ocurre, y por lo tanto hay elementos de  $B$  que no se corresponden con ningún miembro de  $A$ , entonces la función es **no sobreyectiva** (*A into B*).
- Si cada elemento de  $B$  se corresponde con un único elemento de  $A$ , entonces la función es **inyectiva** (*one-to-one*). En caso contrario, si algún miembro de  $B$  al que “llega” más de un elemento de  $A$ , entonces la función es **no inyectiva** (*many-to-one*).

Las funciones que cumplen tanto con la propiedad de ser sobreyectivas como con la de ser inyectivas se denominan **biyectivas**. Estas funciones suelen ser de especial interés dado que su inversa es también una función.

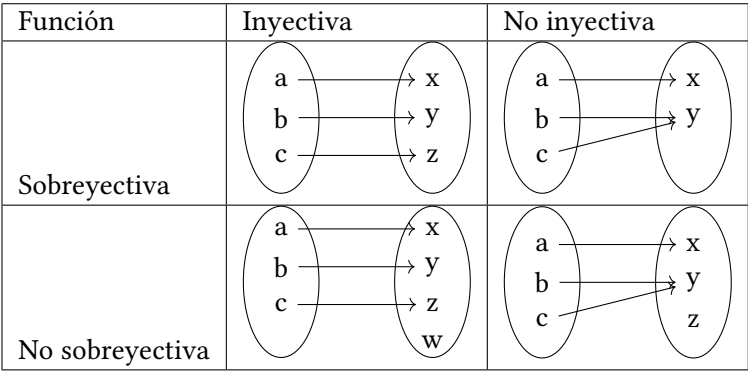


Figure 1.4: Clasificación de funciones

## 1.4 Equivalencia entre funciones y conjuntos

### 1.5 Ejercicios

**Ejercicio 1.1.** Indique cuáles de estos conjuntos son equivalentes y cuáles no:

 Viene de la página 8.

- (24)
- a. {agua, agua carbonatada, gaseosa}
  - b. {soda, gaseosa, H2O}
  - c. {'soda', gaseosa, agua}
  - d. {soda, gaseosa, soda, agua}

Defina el primer conjunto por intensión:



**Ejercicio 1.2.** Existe una regla que dice que los meses que no llevan una 'r' en su nombre son los meses ideales para plantar. Siguiendo esa regla, defina el conjunto de meses aptos para plantar por extensión, por intensión y por reglas recursivas:

## 1. Por extensión



## 2. Por intensión



## 3. Por reglas recursivas



## 1.6 Recapitulación

En este capítulo introdujimos nociones básicas sobre qué significa la formalización en gramática. Observamos que para formalizar es preciso recurrir a un sistema axiomático. Los principales conceptos que se utilizaron a lo largo del capítulo fueron los siguientes:

- **Gramática formal:** Modelo lógico-matemático para generar todas las oraciones gramaticales de una lengua y ninguna de las agramaticales.
- **Función:** Operación matemática que toma elementos de determinado “tipo” y devuelve cierto resultado.
- **Conjunto:** Colección no ordenada de elementos.
- **Equivalencia entre funciones y conjuntos:** Toda función puede representarse en términos de un conjunto (el conjunto de los valores que la función acepta como entrada con sus correspondientes resultados) y todo conjunto puede representarse en términos de una función (la función que devuelve como resultado todos los elementos que pertenezcan al conjunto).

- **Definición de función/conjunto por extensión:** Nombrar todos los elementos que pertenecen a la función o al conjunto en cuestión.
- **Definición de función/conjunto por intensión o abstracción:** Formular una descripción que permita aislar a todos los elementos que pertenecen a la función o conjunto en cuestión.
- **Definición de conjunto por reglas recursivas:** Formular una serie de reglas que permita aislar todos los elementos del conjunto y ninguno que no pertenezca. Esto último se hace a partir de la llamada regla recursiva.

Vimos también las operaciones básicas de conjuntos, como unión, intersección, diferencia, complemento y producto cartesiano.

## 1.7 Lecturas recomendadas

- Barbara Partee et al. 2012. *Mathematical methods in linguistics*. Dordrecht: Kluwer Academics. Capítulo 1 “Basic concepts of set theory”, 3-26; Capítulo 2 “Relations and functions”, 27-37.
- Carlos Peregrín Otero. 1970. *Introducción a la lingüística transformacional*. México DF: Siglo XXI. Parte 2 “Prontuario mínimo del simbolismo a la abstracción”, 75-263.
- Andrés Saab & Fernando Carranza. 2021. *Dimensiones del significado: una introducción a la semántica formal*. Buenos Aires: SADAF. Capítulo 1 “Primeros pasos para una semántica extensional”. 3-38.

## 2 Los lenguajes formales

### 2.1 Introducción

En “La Biblioteca total”, Borges cuenta que si un mono inmortal tipeara al azar en una máquina de escribir durante un tiempo infinito, acabaría produciendo todos los libros que han existido y todos los que puedan existir en el futuro. Así, tarde o temprano tal mono escribiría, por ejemplo, las obras completas de Shakespeare. Ahora bien, supongamos que somos editores y queremos publicar las obras completas de Shakespeare. Sentar a un mono inmortal a escribir infinitamente y publicar todo lo que escriba no sería una buena forma de cumplir nuestro propósito, puesto que nuestra editorial no solo publicaría todas las obras completas de este autor, sino que también incluiría en su catálogo, como dice Borges, tantos fárragos verbales, que varias generaciones perderían su vida tratando de encontrar alguna página tolerable. En otros términos, un mono tecleando al azar no sería un autómata adecuado para el propósito de emular el genio de Shakespeare, porque no basta con que ese autómata genere todas sus obras para cumplir nuestro objetivo, sino que es menester también que no genere ninguna otra cosa. Siguiendo el mismo razonamiento, con seguridad podemos decir que ese mismo mono escribiría todas las oraciones posibles del español. Sin embargo, nadie en su sano juicio podría decir que ese mono sabe español. Es cierto que sus escritos contienen todo lo que se dijo y todo lo que podría llegar a decirse potencialmente en español, pero también es cierto que el catálogo incluye las oraciones de todas las lenguas imaginables e incluso infinidad de cadenas que no pertenecen a ninguna lengua humana posible. El mono, cuyo ingenio se limita al azar, sería incapaz de diferenciar todas esas cadenas. En conclusión, frente al propósito de construir un autómata que hable español, ese mono sería inadecuado. Si quisiéramos restringir el poder expresivo del mono de modo tal que solo hable español, tendríamos que dotarlo de una serie de instrucciones de tal naturaleza que su tecleo solo produjera las combinaciones que den lugar a todas las cadenas del español y a ninguna que no pertenezca al español. Así, para poder diseñar esas instrucciones, resulta útil pensar el español como un conjunto de cadenas, en el sentido de la teoría de conjuntos: el conjunto de todas sucesiones de teclas que dan lugar a cadenas del español y de ninguna que dé lugar a cadenas que no sean del español.

Diremos que una lengua, entendida en estos términos, constituye un lenguaje. Podemos concebir un lenguaje, entonces, como cualquier conjunto de cadenas que queramos que el mono escriba y que excluya todo el conjunto de cadenas que queramos que el mono soslaye. Los lenguajes, entendidos en este sentido, son de gran interés para la llamada lingüística matemática o algebraica y para las ciencias de la computación. El resto del capítulo está destinado a indagar en este concepto.

### 2.2 Alfabetos, cadenas y lenguajes

Martinet sostiene que el lenguaje humano está sujeto a dos articulaciones:

La primera articulación del lenguaje es aquella con arreglo a la cual todo hecho de experiencia que se vaya a transmitir, toda necesidad que se desee hacer conocer a otra persona, se analiza en una sucesión de unidades, dotadas cada una de una forma vocal y de un sentido. (Martinet 1984 [1960]: 22)

Cada una de estas unidades de la primera articulación presenta, como hemos visto, un sentido y una forma vocal (o fónica). Pero no puede ser analizada en unidades sucesivas más pequeñas dotadas de sentido. El conjunto *cabeza* quiere decir “cabeza” y no se puede atribuir a *ca-*, *-be-*, *-za*, sentidos distintos cuya suma sea equivalente a “cabeza”. Pero la forma vocal es analizable en una sucesión de unidades, cada una de las cuales contribuye a distinguir *cabeza* de otras unidades como *cabete*, *majeza* o *careza*. Es a esto a lo que se designará como la segunda articulación del lenguaje.

(Martinet 1984 [1960]: 24)

En términos más familiares, podemos decir que la primera articulación se corresponde con los ítems léxicos. La naturaleza de la segunda articulación dependerá de la modalidad de verbalización, que puede ser, en principio, oral o escrita:

En el hablar corriente, ‘el lenguaje’ designa propiamente la facultad que tienen los hombres de entenderse por medio de signos vocales. Merece la pena detenerse en este carácter vocal del lenguaje. En los países civilizados, desde hace algunos milenios se hace uso con mucha frecuencia de signos pictóricos o gráficos que corresponden a los signos vocales del lenguaje. Esto es lo que se llama escritura. Hasta la invención del fonógrafo, todo signo vocal emitido era percibido inmediatamente o quedaba perdido para

siempre. Por el contrario, un signo escrito, duraba tanto cuanto durara su soporte: piedra, pergamino o papel, y los rasgos dejados sobre este soporte por el buril, el estilo o la pluma.

(Martinet 1991 [1960]: 14)

En la modalidad oral, entonces, las unidades relevantes son los llamados fonemas, esto es, los sonidos distintivos que utiliza cada lengua. En la modalidad escrita para una lengua con escritura alfabética, las unidades relevantes serán los grafemas. En una computadora, las unidades relevantes serán los caracteres, que pueden pertenecer a distintas fuentes o *encodings*: UTF-8, ASCII, etc.. Denominamos *alfabeto* al conjunto no vacío de símbolos que constituya la segunda articulación del lenguaje (independientemente de su modalidad): grafemas, fonos, caracteres. El alfabeto se designa convencionalmente con la letra  $\Sigma$ . En (1) se ilustran algunos ejemplos de alfabetos posibles:

- (1) a. Alfabeto Latino =  $\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$
- b. Dígitos =  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- c. Dígitos para código binario =  $\{0, 1\}$

La secuencia de símbolos de un alfabeto  $\Sigma$ , ya sean iguales o diferentes, se conoce con el nombre de *cadena*. Así, son cadenas las palabras, las oraciones gramaticales, los constituyentes, pero también las no palabras, las oraciones agramaticales, las expresiones que no conforman constituyente, etc.

- (2) a. Si  $\Sigma = \{0, 1\}$ ,  $\{0, 1, 01, 10, \dots\}$  serán posibles cadenas.
- b. Si  $\Sigma = \{a, b, c, \dots, z\}$ ,  $\{a, ba, casa, \dots\}$  serán posibles cadenas.

La *longitud de una cadena*  $w$ , denotada  $|w|$ , indica la cantidad de unidades mínimas contenidas en la cadena, es decir, la cantidad de veces que se tomó un símbolo del alfabeto para construir la secuencias. La única cadena cuya longitud es 0 es la *cadena vacía*, denotada  $\epsilon$ .

- (3) a.  $|\text{barco}| = 5$
- b.  $|\text{casa}| = 4$
- c.  $|\text{aa}| = 2$
- d.  $|\epsilon| = 0$

## 2 Los lenguajes formales

Todas las cadenas de determinada longitud  $k$  que se pueden construir con un alfabeto  $\Sigma$  se representan convencionalmente  $\Sigma^k$ .

Por ejemplo, dado el alfabeto  $\Sigma = \{a, b\}$ , se dan las siguientes extensiones:

- (4) a.  $\Sigma^0 = \{\emptyset\}$   
 b.  $\Sigma^1 = \{a, b\}$   
 c.  $\Sigma^2 = \{aa, ab, ba, bb\}$   
 d.  $\Sigma^3 = \{aaa, aab, abb, aba, bbb, bba, baa, bab\}...$

La *clausura de Kleene* es una operación que, al aplicarse a un alfabeto  $\Sigma$ , nos permite obtener el conjunto de todas las cadenas que se pueden obtener a partir de él. Para esta operación se usa la notación  $\Sigma^*$ . En términos de teoría de conjuntos,  $\Sigma^* = \{\Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \Sigma^4 \cup \dots\}$ . Obsérvese que este conjunto ( $\Sigma^*$ ) es siempre infinito.

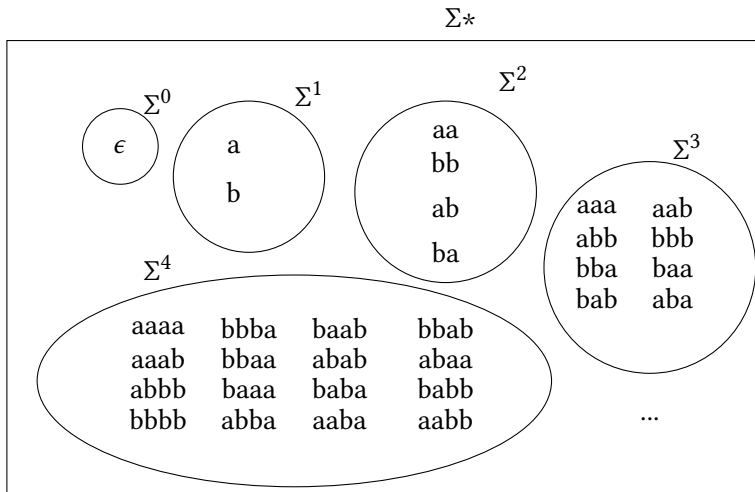


Figure 2.1: Imagen de ejemplo de las cadenas posibles generadas por un alfabeto  $\Sigma = \{a, b\}$

Si contamos la *u* con diéresis y las vocales acentuadas como caracteres distintos de las no acentuadas, el español tiene 33 caracteres (solo minúsculas y sin contar signos de puntuación). Supongamos que estos 33 caracteres conforman el alfabeto  $\Sigma$ . Ahora bien,  $\Sigma^*$  incluye una infinita cantidad de cadenas que no forman parte del español, como por ejemplo *dkfjhg* o *tuqpeigh*



Un **lenguaje L** es un conjunto de cadenas *particularmente relevante* que está incluido en  $\Sigma^*$ .

Al ser un conjunto, un lenguaje puede definirse, como vimos anteriormente, por intensión o por extensión. Pero la segunda forma resulta inviable dado que los lenguajes están constituidos por infinitos elementos. Es por eso que se prefiere su definición intensional.

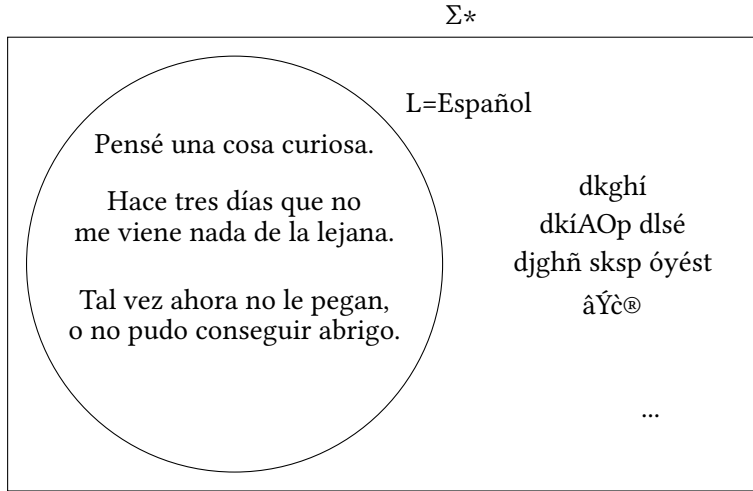


Figure 2.2: Lenguaje español como subconjunto de  $\Sigma^*$  para el alfabeto  $\Sigma = \text{UTF-8}$

## 2.3 Operaciones

### 2.3.1 Operaciones entre cadenas

Existen ciertas operaciones que se puede establecer entre cadenas. En primer lugar, la *concatenación* de dos cadenas  $w_1$  y  $w_2$  se denota  $w_1 \cdot w_2$  y consiste en unir una cadena a otra de manera tal que las unidades que forman a cada una queden dispuestas en una única secuencia. Si  $w_1 = \langle x_1, x_2, \dots, x_n \rangle$  y  $w_2 = \langle y_1, y_2, \dots, y_n \rangle$ , entonces  $w_1 \cdot w_2 = \langle x_1, \dots, x_n, y_1, \dots, y_n \rangle$ . En un ejemplo más concreto, si  $w_1 = \text{limpia}$  y  $w_2 = \text{parabrisas}$ , entonces  $w_1 \cdot w_2 = \text{limpiaparabrisas}$ . La longitud de la concatenación de dos cadenas es el resultado de la suma de sendas longitudes. Más formalmente:  $|w_1 \cdot w_2| = |w_1| + |w_2|$ .


El operador *exponente* nos permite concatenar una cadena consigo misma tantas veces como dicho operador indique. Se denota  $w^n$ , donde  $w$  es la cadena que

## 2 Los lenguajes formales

sufre la operación y  $n$  es la cantidad de veces que se la debe concatenar. Cuando  $n = 0$  ( $w^0$ ), el resultado es una cadena vacía ( $\epsilon$ ). En los casos en los que  $n > 0$ ,  $w^n = w^{n-1} \cdot w$ . En particular,  $w^1 = w$ . En un ejemplo más concreto, si  $w = \text{hola}$ , entonces se da a lugar a las siguientes equivalencias:

- (5) a.  $w^0 = \epsilon$
- b.  $w^1 = \text{hola}$
- c.  $w^2 = w^1 \cdot w = \text{holahola}$
- d.  $w^3 = w^2 \cdot w = \text{holaholahola}$ .

La operación *reversa* de una cadena  $w$  se denota  $w^R$  y consiste en ordenar las unidades que conforman a  $w$  de manera inversa. Si  $w = \langle x_1, x_2, \dots, x_n \rangle$ , entonces  $w^R = \langle x_n, \dots, x_2, x_1 \rangle$ . Por ejemplo, si  $w = \text{luz}$ , entonces  $w^R = \text{zul}$ .

 Ver ejercicios en [2.1](#).


### 2.3.2 Operaciones entre lenguajes

Así como hay operaciones entre conjuntos, también las hay, naturalmente, entre lenguajes, puesto que estos son en última instancia conjuntos. Algunas de las más usuales son las siguientes. En primer lugar, si  $L$  es un lenguaje, el reverso de  $L$  ( $L^R$ ) también es un lenguaje, formado por todas las cadenas que se obtienen al aplicar la operación de reverso a cada una de las cadenas de  $L$ .

En segundo lugar, si  $L_1$  y  $L_2$  son lenguajes, la concatenación entre ellos ( $L_1 \cdot L_2$ ) es un lenguaje formado por la concatenación de cada cadena del primero con cada cadena del segundo.

En tercer lugar, si  $L$  es un lenguaje  $\{\alpha, \beta\}$ ,  $L^0 = \{\epsilon\}$ ,  $L^1 = \{\alpha, \beta\}$ ,  $L^2 = \{\alpha\alpha, \alpha\beta, \beta\beta, \beta\alpha\}$ .

En cuarto lugar, si  $L$  es un lenguaje, la concatenación potencialmente infinita de cada cadena de  $L$  ( $L^*$ ) también es un lenguaje. Basta que  $L$  tenga un solo elemento para que  $L^*$  sea infinito. Esto se conoce como la clausura de Kleene.

 Ver ejercicios en [2.2](#).

## 2.4 Lenguajes como problemas

La noción de lenguaje se extiende no solamente a los lenguajes naturales sino también a cualquier conjunto de cadenas formadas a partir de un alfabeto  $\Sigma$ .

Dentro de la teoría de autómatas, plantear un problema significa preguntarse si un elemento dado pertenece a determinado lenguaje o no ([Hopcroft et al. 2006](#):

31). Esto significa, dada una cadena  $w \in \Sigma^*$ , decidir si dicha cadena pertenece o no a un  $L$  determinado. En consecuencia, existe una equivalencia entre lenguajes y problemas. Por supuesto, no cualquier problema humanamente concebible puede transformarse en términos de un lenguaje. Que

## 2.5 Sobre la relevancia lingüística de los lenguajes


En inglés, *language* abarca tanto a *lengua* como a *lenguaje*. Esto hace que a la hora de traducir esta palabra, se presente cierta oscilación. En este libro, utilizamos *lenguaje*, como aclaramos previamente, para definir a un conjunto de cadenas. Este concepto se halla presente ya en Chomsky 1957, aunque la traducción de Peregrín Otero opta por la palabra *lengua*: “En adelante entenderé que una *lengua* es un conjunto (finito o infinito) de oraciones, cada una de ellas de una longitud finita y construida a partir de un conjunto de elementos finito” (Chomsky 1957: 27). Sin embargo, unos años después, Chomsky abandona este concepto y lo reemplaza por el de Lengua-I. La lengua-I es el sistema intensional que posee cada hablante y que permite producir todas las oraciones gramaticales de una lengua y ninguna de las agramaticales. Este cambio terminológico no es injustificado. Chomsky sostiene que la lingüística es una rama de la biología. El objeto de estudio, por lo tanto, tiene que ser una entidad que tenga realidad biológica. La lengua-I la tiene, puesto que es, en última instancia, un órgano mental. El lenguaje, en cambio, es un conjunto de cadenas y, por lo tanto, se trata de un objeto abstracto, no de un objeto biológico. Como tal, no tiene lugar en el paradigma chomskyano. Chomsky 1990 equipara la noción de lenguaje con la de Lengua-E y argumenta que este constructo teórico tiene sentido en el estudio de los lenguajes formales, pero no lo tiene en la lingüística formal.

Por el contrario, no hay ningún inconveniente en asumir el concepto de lenguaje si se considera que la lingüística es una rama de las matemáticas. Por este motivo, la noción de lenguaje, actualmente forma parte de la rama de la lingüística algebraica o lingüística matemática, es decir, “el estudio lógico-matemático de ciertos lenguajes ideales que podrían eventualmente caracterizar las lenguas naturales” (Quesada 1974: 29).

👉 Ver ejercicio en 2.3.

## 2.6 Ejercicios

### Ejercicio 2.1.

 Viene de página 24.

Escriba el resultado de aplicar las siguientes operaciones a estas cadenas:


1. pisa · papel =



2. severlaatseotse<sup>R</sup> =



### Ejercicio 2.2. Operaciones sobre lenguajes

 Viene de página 24.

Dados los siguientes lenguajes, escriba el resultado de aplicar la operación relevante en cada caso.

- (6) a.  $L_1 = \{\text{hola, chau, buenas}\}$   
b.  $L_2 = \{a, b\}$

1.  $L_2 \cup L_1 =$

2.  $L_1 \cap L_2 =$

3.  $L_1^R =$


4.  $L_1^1 =$

5.  $L_2 \cdot L_1 =$

6.  $L_2^3 =$

7.  $L_2^R =$

## Ejercicio 2.3.

 Viene de página 25.

Identifique cuál de estos lenguajes caracteriza mejor las cadenas formadas por la concatenación de las subcadenas subrayadas y justifique<sup>1</sup>.

- (7) a. Catalina y Martín tomaron respectivamente un Campari y un Fernet.  
 b. Catalina, Martín y Federico tomaron respectivamente un Campari, un Fernet y una cerveza.  
 c. Catalina, Martín, Federico y Victoria tomaron respectivamente un Campari, un Fernet, una cerveza y un aperitivo.

- $a^n b^n$
- $(ab)^+$
- $a^* b^*$

- (8) a. El mono  
 b. El mono adorable

<sup>1</sup>Para una aplicación práctica de este tipo de ejercicio, ver Yang & Piantadosi 2022. Entiéndase por  $a\{n\}b\{n\}$  una cadena de  $n$  ocurrencias de  $a$  seguida de  $n$  ocurrencias de  $b$ .

## 2 Los lenguajes formales

- c. El mono adorable y amistoso
  - d. El mono adorable, amistoso y joven.
- $a^n b^n$
  - $a^*$
  - $(ab)^+$
  - $a^* b^*$
- (9)
- a. En la fiesta de ayer me tomé un fernet.
  - b. En la fiesta de ayer me tomé un fernet y un campari.
  - c. En la fiesta de ayer me tomé un fernet, un campari y una cerveza.
  - d. En la fiesta de ayer me tomé un fernet, un campari, una cerveza y un aperitivo.
- $ab^n cd^n$
  - $a^*$
  - $(ab)^+$
  - $a^* b^*$
- (10)
- a. No creo que que sea providencial deje de sorprenderme.
  - b. No creo que que que sea providencial deje de sorprenderme deje de sorprenderme.
  - c. No creo que que que sea providencial deje de sorprenderme deje de sorprenderme deje de sorprenderme
- $ab^n cd^n$

- $a^*$
- $(ab)^+$
- $a^*b^*$

## 2.7 Recapitulación

En este capítulo desarrollamos el concepto de lenguaje como conjunto de cadenas definidas a partir de un alfabeto  $\Sigma$ . Hemos visto que esta es la forma en la que se conciben las lenguas en la lingüística matemática. Así como los conjuntos están sujetos a distintas operaciones, también las cadenas y los lenguajes admiten distintas operaciones: la reversa, la concatenación, la exponenciación y la clausura de Kleene.

## 2.8 Lecturas recomendadas

- 📖 John E Hopcroft et al. 2006. *Automata theory, languages, and computation*. Boston, Massachusetts: Addison-Wesley. Capítulo 1 “Automata: The Methods and the Madness”, pp. 1-35.





## 3 Jerarquía de los lenguajes formales

### 3.1 Introducción

En el capítulo anterior hemos visto que un lenguaje es un conjunto de cadenas definida a partir de un determinado alfabeto. Ahora bien, supongamos el alfabeto  $\Sigma = \{0, 1\}$ . Con este alfabeto de base es posible definir, entre muchos otros, lenguajes tan diversos como los siguientes:.

- (1) a. L1: cualquier número de unos y ceros en cualquier orden
- b. L2: un número cualquiera de unos seguidos de un número cualquiera de ceros
- c. L3: un número cualquiera de unos seguidos del mismo número de ceros.
- d. L4: un número de unos equivalente al cuadrado del número de ceros que haya
- e. L5: una secuencia de ceros y unos que conforman un programa que hace operaciones con cadenas de ceros y unos y que se acepta a sí misma como entrada.

Un rápido examen nos sugiere que estos lenguajes difieren con respecto a la complejidad de la formulación de la descripción que nos permite obtenerlos. A grandes rasgos, esta complejidad intuitiva tiende a tener un correlato también en la definición por intensión o abstracción que propongamos para describirlos. Esta complejidad no es gratuita: a mayor complejidad, se requiere un algoritmo que permita formular restricciones más finas y esto acarrea mayor costo de procesamiento (en tiempo y/o espacio/memoria). Es posible relacionar, entonces, la complejidad con el poder discriminatorio: un lenguaje cuya descripción incluya mayores restricciones para discriminar qué cadenas le pertenecen y cuáles no le pertenecen es más complejo. Los lenguajes se caracterizan por su poder discriminatorio en distintas clases. Estas clases se denominan con el símbolo  $\mathcal{L}$  y conforman la siguiente jerarquía desde el más simple al más complejo (desde ya, estas clases están sujetas a subdivisiones internas):

- (2) a. Lenguajes regulares

### 3 Jerarquía de los lenguajes formales

- b. Lenguajes independientes de contexto
- c. Lenguajes sensibles al contexto
- d. Lenguajes irrestrictos

Esto es lo que se conoce popularmente como *Jerarquía de lenguajes formales* o *Jerarquía de Chomsky*. Este capítulo está destinado a abordar esta jerarquía.

## 3.2 Jerarquía de lenguajes formales

La jerarquía de Chomsky se define en términos del poder expresivo de los distintos lenguajes. Si los lenguajes no difirieran en este aspecto, no podrían ser jerarquizados. Es importante destacar que el poder expresivo no está relacionado con la cantidad de cadenas que se puedan generar, sino con la capacidad de discriminar cadenas. Por ejemplo, el lenguaje 6 en (3), genera para cualquier alfabeto  $\Sigma$  que tenga al menos un elemento un número infinito de cadenas y, obviamente, cualquier cadena concebible con ese alfabeto va a pertenecer a ese lenguaje.

- (3) L6: cualquier número  $n$  mayor o igual a 0 de ocurrencias de cualquiera de las letras del alfabeto  $\Sigma$ .

El hecho de que L6 pueda generar cualquier cadena concebible podría crearnos la ilusión de que se trata de un lenguaje muy poderoso. Sin embargo, el lenguaje L6 pertenece a la clase más simple de lenguajes, los lenguajes regulares. Esto se explica porque la pertenencia a una clase de lenguajes depende de la posibilidad para discriminar cadenas antes que para generarlas. Este lenguaje es muy simple porque básicamente no impone ninguna restricción a la combinación de símbolos del alfabeto  $\Sigma$ . En cambio, supongamos el lenguaje L2 que repetimos a continuación:

- (4) L3: un número cualquiera de unos seguidos del mismo número de ceros.

La jerarquía de lenguajes formales está definida en términos de inclusión:

- (5) Lenguajes regulares  $\subset$  Lenguajes independientes de contexto  $\subset$  Lenguajes sensibles al contexto  $\subset$  Lenguajes irrestrictos.

Una clase de lenguaje  $\mathcal{L}$  es cerrada bajo cierta operación si al aplicar esa operación a lenguajes de esa clase se obtienen lenguajes de la misma clase  $\mathcal{L}$ . Por ejemplo, se sabe que si  $L$  es un lenguaje regular, sea cual fuera,  $L^R$  también es un

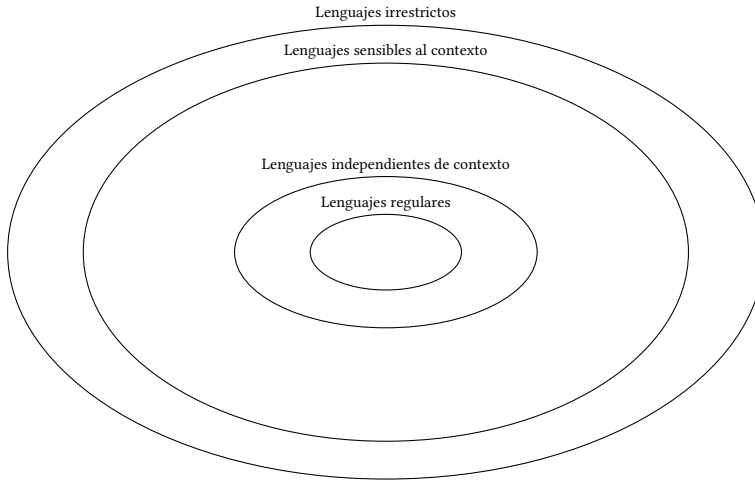


Figure 3.1: Diagrama de Venn de la jerarquía de lenguajes formales

lenguaje regular. Se puede concluir entonces que la clase de lenguajes regulares es cerrado bajo la operación reversa.

Un problema sumamente relevante que involucra la relación entre lenguajes y cadenas es el problema de pertenencia, que tiene dos grandes vertientes. En primer lugar, el *problema de pertenencia a secas* o *fixed language recognition problem* se pregunta, dada una cadena  $w$  y un lenguaje  $L$ , si  $w \in L$ . En segundo lugar, el *problema de reconocimiento universal* se pregunta, dada una gramática  $G$  y una cadena  $L$ , si  $w \in L(G)$ , es decir, si  $w$  pertenece al lenguaje generado por  $G$ . Como es de esperar, el problema del reconocimiento universal es el que tiene mayor peso en la teoría lingüística, ya que es el que involucra el trabajo con gramáticas formales (Ristad 1986). Este problema se aborda computacionalmente con dos clases de algoritmos. Los *algoritmos de reconocimiento* son algoritmos que chequean si determinada oración se sigue como teorema de determinada gramática (i.e., si determinada cadena pertenece al lenguaje generado por esa gramática). Por su parte, los *algoritmos de parsing*, además de chequear si determinada oración se sigue como teorema de determinada gramática, devuelven, en caso positivo, una estructura para esa oración.

### 3.3 Formas de definir lenguajes

Como todo conjunto, los lenguajes se pueden definir por extensión o por intención. No obstante, puesto que la definición por extensión tiene la limitación de

### 3 Jerarquía de los lenguajes formales

que solo es viable para conjuntos finitos que tengan un número medianamente reducido de miembros, generalmente se opta por la definición por abstracción. Convencionalmente, existen dos grandes formas de definir lenguajes por intención: los autómatas y las gramáticas (formales).

#### 3.3.1 Autómatas

Los autómatas son matemáticamente tuplas que consisten mínimamente de los siguientes elementos:

- $\sigma$  = Alfabeto
- $Q$  = Conjunto de Estados
- $q_0$  = un único estado inicial
- $F$  = un conjunto de estados finales
- $\delta$  = un conjunto de instrucciones que especifican qué hace el autómata en determinado estado ante determinado símbolo y a qué estado pasa.

Como son objetos matemáticos, los autómatas no tienen realidad física. No son máquinas que puedan romperse o a las que haya que proveer de corriente. En su lugar, se trata de mecanismos abstractos. Normalmente, se los suele representar mediante grafos. Supongamos un  $L_6$  definido del siguiente modo:

- (6) a.  $L_6 = \{w: w \text{ es una cadena obtenida por la concatenación de } a \text{ con } n \text{ número de ocurrencias de } b \text{ para cualquier } n \geq 0\}$   
b.  $L_6 = \{a, ab, abb, abbb, abbbb...\}$

Se puede ver una posible representación mediante un grafo para el autómata  $A_6$  que describe el lenguaje  $L_6$  en la figura 3.2

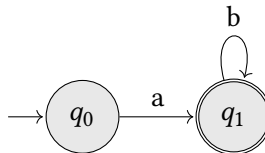


Figure 3.2: Autómata para  $A_6$


Los autómatas se pueden representar también mediante tablas de transición en las que se especifica qué hace cada estado (representado en las filas) ante cada

Table 3.1: Tabla de transición para A6

	a	b
$\rightarrow q_0$	$q_1$	$\emptyset$
$*q_1$	$\emptyset$	$q_1$

símbolo (representado en las columnas). Por ejemplo, en la de la tabla 3.1 se ve la representación mediante una tabla de transición del autómata A6.

En esta tabla se especifica que  $q_0$  es el estado inicial (esto se representa por la flecha), que en ese estado, ante el símbolo a el autómata pasa al estado  $q_1$  y que si, en su lugar, se encuentra con el símbolo b, la máquina se detiene sin llegar a un estado de aceptación. Con respecto al estado  $q_1$ , se indica mediante el asterisco que es un estado final y que ese estado rechazará la cadena si se encuentra con a o que eventualmente seguirá en el estado  $q_1$ , hasta que acepte la cadena, si esta se termina.

 Ver ejercicio 3.5.1.

En (7) se define un autómata para este lenguaje L6 expresado como una quintupla.

- (7) a.  $A_6 = (\{a, b\}, \{q_0, q_1\}, q_0, \{q_1\}, \{ \langle q_0, a, q_1 \rangle, \langle q_0, b, \emptyset \rangle, \langle q_1, a, \emptyset \rangle, \langle q_1, b, q_1 \rangle \})$
- b.  $A_6 = (\Sigma, Q, q_0, F, \delta)$  donde  $\Sigma = \{a, b\}$ ,  $Q = \{q_0, q_1\}$ ,  $q_0, \{q_1\}$ ,  $F = \{q_1\}$  y  $\delta = \{ \langle q_0, a, q_1 \rangle, \langle q_0, b, \emptyset \rangle, \langle q_1, a, \emptyset \rangle, \langle q_1, b, q_1 \rangle \}$

El poder discriminativo de los autómatas depende fundamentalmente de la naturaleza de las instrucciones que especifican lo que hace el autómata. El autómata ejemplificado arriba es del tipo más sencillo, que ante cada símbolo solo es capaz de rechazarlo (ir a  $\emptyset$ ) o aceptarlo y pasar a otro estado  $q_i$ . Otros autómatas son capaces de alterar los símbolos que ven como entrada, regresar hacia atrás en la cadena y alterar los símbolos previos e incluso extender o reducir la cadena. Según la complejidad de sus instrucciones los autómatas se clasifican en distintos tipos que son capaces de reconocer o generar los siguientes tipos de lenguajes, según se especifica en la tabla 3.2.

Los autómatas tienen la característica de que solo tienen generación débil, en la terminología de Chomsky 1963. Es decir, son capaces de generar cadenas, pero no son capaces de asignarles una estructura. Por este motivo, en lingüística formal, los autómatas tienen una importancia reducida.

### 3 Jerarquía de los lenguajes formales

Table 3.2: Equivalencia entre lenguajes y autómatas

Lenguajes	Autómatas
Lenguajes regulares	Autómatas de estados finitos
Lenguajes independientes de contexto	Autómatas de pila
Lenguajes sensibles al contexto	Autómatas linealmente acotados
Lenguajes irrestrictos	Máquinas de Turing

#### 3.3.2 Las gramáticas

Las gramáticas son sistemas deductivos formados por axiomas, es decir, ciertas proposiciones básicas que se dan por supuestas de antemano y que no se demuestran, y reglas de deducción que permiten derivar las oraciones del lenguaje natural como teoremas a partir de los axiomas. Como se verá en las siguientes tres partes de este libro, existen distintas estrategias que pueden seguir las gramáticas respecto de cómo conciben las reglas de deducción, pero todas comparten esta organización general y poseen, además de generación débil, es decir, la posibilidad de generar cadenas, generación fuerte, es decir, la posibilidad de generar estructuras para esas cadenas.

Formalmente las gramática más típicas se definen como una cuádrupla. Si bien en lógica se suele anotar las listas con corchetes angulares, en lingüística matemática las gramáticas entendidas como listas se suelen escribir con paréntesis. De este modo, una gramática se define del siguiente modo:

$$(8) \quad G = (V_T, V_N, S, R)$$

Cada uno de estos elementos que conforman la cuádrupla se definen del siguiente modo:

- (9)    a.  $V_T$  = símbolos terminales  
      b.  $V_N$  = símbolos no terminales  
      c.  $S$  = el axioma inicial, incluido en  $V_N$   
      d.  $R$  = conjunto de reglas

Si las gramáticas poseen reglas de reescritura que limitan el lado izquierdo solo a un símbolo no terminal, es posible a partir de ellas generar un árbol de constituyentes.

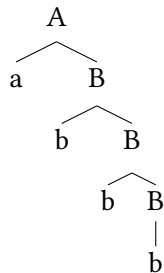
(10) Gramática 1 para L6

- a.  $V_T$  = símbolos terminales = {a, b}
- b.  $V_N$  = símbolos no terminales = {A, B}
- c. S = el axioma inicial, incluido en  $V_N = A$
- d. R = conjunto de reglas =  $\left\{ \begin{array}{l} R1. A \rightarrow aB \\ R2. B \rightarrow bB \\ R3. B \rightarrow b \end{array} \right\}$

(11) Derivación para abbb

- a. aB por R1
- b. abB por R2
- c. abbB por R2
- d. abbb por R3

(12) Estructura para abbb



Las gramáticas tienen dos clases de capacidades generativas, que se postularon inicialmente en Chomsky 1963 y Chomsky & Miller 1963.

- **Capacidad generativa débil:** La capacidad de una gramática de generar cadenas (i.e. oraciones).
- **Capacidad generativa fuerte:** La capacidad de una gramática de generar estructura para sus cadenas.

Así como los autómatas se clasifican según el tipo de clases de lenguajes que pueden generar, lo mismo ocurre con las gramáticas. Según su capacidad generativa débil, autómatas y gramáticas se clasifican en los tipos que se especifican en la tabla 3.3.

### 3 Jerarquía de los lenguajes formales

Table 3.3: Equivalencia entre lenguajes, gramáticas y autómatas

Lenguajes	Gramáticas	Autómatas
Lenguajes regulares	Gramáticas regulares	Autómatas de estados finitos
Lenguajes independientes de contexto	Gramáticas independientes de contexto	Autómatas de pila
Lenguajes sensibles al contexto	Gramáticas sensibles al contexto	Autómata linealmente acotado
Lenguajes irrestrictos	Gramáticas irrestrictas	Máquina de Turing

## 3.4 Clases de lenguajes

### 3.4.1 Clases de lenguajes por su capacidad discriminatoria

#### 3.4.1.1 Lenguajes regulares

##### 3.4.1.1.1 Expresiones regulares

Los lenguajes regulares son aquellos conjuntos de cadenas que pueden ser descriptos mediante expresiones regulares.


- (13) Dado un alfabeto  $\Sigma$ , el conjunto de expresiones regulares sobre  $\Sigma$  se define de la siguiente forma:
- $\emptyset$  es una expresión regular
  - $\epsilon$  es una expresión regular
  - Si  $a \in \Sigma$ ,  $a$  es una expresión regular
  - Si  $r_1$  y  $r_2$  son expresiones regulares, también lo son  $(r_1 + r_2)$  y  $(r_1 \cdot r_2)$
  - Si  $r$  es una expresión regular, también lo es  $(r)^*$
  - Nada más es una expresión regular

Los corchetes dobles  $\llbracket \rrbracket$  se utilizan para expresar la función denotación, que es aquella función que toma un argumento y devuelve como resultado su denotación. En el caso de las expresiones regulares, si  $r$  es una expresión regular,  $\llbracket r \rrbracket$  es el conjunto de cadenas que  $r$  describe.

- (14) Dado un alfabeto  $\Sigma$ , el conjunto de expresiones regulares sobre  $\Sigma$  se define de la siguiente forma:



- a.  $\llbracket \emptyset \rrbracket = \{\}$
- b.  $\llbracket \epsilon \rrbracket = \{\epsilon\}$
- c. Si  $a \in \Sigma$ ,  $\llbracket a \rrbracket = \{a\}$
- d. Si  $r_1$  y  $r_2$  son expresiones regulares cuyas denotaciones son  $\llbracket r_1 \rrbracket$  y  $\llbracket r_2 \rrbracket$ ,  $\llbracket (r_1 + r_2) \rrbracket = \llbracket r_1 \rrbracket \cup \llbracket r_2 \rrbracket$  y  $\llbracket (r_1 \cdot r_2) \rrbracket = \llbracket r_1 \rrbracket \cdot \llbracket r_2 \rrbracket$
- e. Si  $r$  es una expresión regular,  $\llbracket (r)^* \rrbracket = \llbracket r \rrbracket^*$

 Ver ejercicio en 3.5.2.

La notación de arriba no es usual en la práctica. En la tabla 3.4 se especifican algunas de las convenciones notacionales que utiliza la librería re de Python.

Símbolo	Significado	Ejemplo
*	cero o más ocurrencias del caracter o bloque anterior	$\llbracket a^* \rrbracket = \{w \mid w \text{ es una cadena formada por la concatenación de cero o más ocurrencias de } a\}$
+	una o más ocurrencias del caracter o bloque anterior	$\llbracket a^+ \rrbracket = \{w \mid w \text{ es una cadena formada por la concatenación de una o más ocurrencias de } a\}$
?	una o ninguna ocurrencia del caracter o bloque anterior	$\llbracket ab? \rrbracket = \{a, ab\}$
.	cualquier caracter excepto cambio de línea	$\llbracket a. \rrbracket = \{aa, ab, ac, ad, \dots\}$
{n}	n cantidad de ocurrencias del caracter o bloque anterior	$\llbracket a\{3\} \rrbracket = \{aaa\}$
{n,m}	entre n y m cantidad de ocurrencias del caracter o bloque anterior (cuando se usa para matchear, matchea con el menor número posible)	$\llbracket a\{3,6\} \rrbracket = \{aaa, aaaa, aaaaa, aaaaaa\}$

### 3 Jerarquía de los lenguajes formales

Símbolo	Significado	Ejemplo
<code>[]</code>	disyunción entre lo que aparezca adentro	<code>[[abcd]] = {a,b,c,d}</code>
<code>[x-y]</code>	rango de caracteres que aparecen entre x e y	<code>[[a-d]] = {a,b,c,d}</code>

Table 3.4: Convenciones notacionales de expresiones regulares siguiendo la librería re de Python (<https://docs.python.org/3/library/re.html>)

Asimismo, las convenciones de las expresiones regulares abarcan también un conjunto de abreviaturas. En la tabla 3.5 se resumen algunas de las más importantes.

Table 3.5: Abreviaturas para expresiones regulares siguiendo la librería re de Python (<https://docs.python.org/3/library/re.html>)

Abreviatura	Significado
<code>\s</code>	espacio en blanco
<code>\S</code>	cualquier caracter que no sea un espacio en blanco
<code>\d</code>	cualquier dígito. Equivale a <code>[0-9]</code>
<code>\D</code>	cualquier símbolo que no sea un dígito
<code>\w</code>	cualquier caracter alfanumérico

Los lenguajes regulares son cerrados ante las siguientes operaciones:

1. unión
2. concatenación
3. clausura de Kleene

#### 3.4.1.1.2 Autómatas de estados finitos

Además de expresiones regulares, los lenguajes regulares se pueden reconocer también mediante autómatas de estados finitos, esto es, autómatas que recorren una cinta y rechazan o aceptan sus símbolos de acuerdo a lo que determine el estado en el que se encuentra el autómata. Un ejemplo de un autómata finito es el de la figura 3.3.

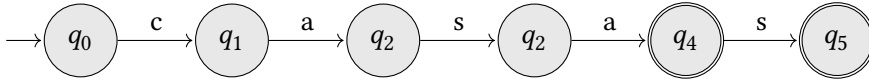


Figure 3.3: Autómata finito que reconoce el lenguaje {casa, casas}

### 3.4.1.1.3 Gramáticas regulares

👉 Ver ejercicio en 3.5.3.

Alternativamente, los lenguajes regulares pueden ser generados o reconocidos mediante gramáticas regulares, es decir, mediante gramáticas que restringen las reglas disponibles a los dos tipos en (??), en donde X o Y deben leerse como símbolos no terminales (sean distintos o no) y x como un símbolo terminal:

- (15) a.  $X \rightarrow x Y$   
b.  $X \rightarrow x$

Por ejemplo

- (16) Gramática regular equivalente a la expresión regular  $a^+$

- a. R1.  $A \rightarrow a A$   
b. R2.  $A \rightarrow a$

- (17) Derivación para  $aaaa$

- a.  $aA$

Por R1

- b.  $aaA$

Por R1

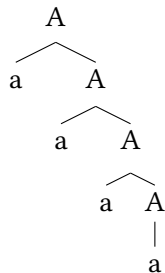
- c.  $aaaA$

Por R1

- d.  $aaaa$

Por R2

- (18) Árbol para la cadena  $aaaa$



#### 3.4.1.1.4 Transducers

Las expresiones regulares se usan en el procesamiento del lenguaje natural para hacer búsquedas o, por ejemplo, para armar diccionarios. Hemos visto que las expresiones regulares caracterizan lenguajes, esto es conjuntos. Sabemos que dados dos conjuntos (idénticos o diferentes) podemos construir pares ordenados (*i.e.*, relaciones) formados por elementos del primer conjunto y elementos del segundo. Un tipo particular de relación es lo que se conoce como relación regular, que establece equivalencias entre lenguajes regulares. Los autómatas que se encargan de caracterizar este tipo de relaciones se conocen con el nombre de *transducers*. Ejemplos de aplicaciones concretas que usan relaciones regulares en el ámbito del PLN son las siguientes:

- **POStagging:** Algoritmos que definen pares ordenados de palabras y sus respectivas etiquetas categoriales.
- **Lemmatizing:** Algoritmos que definen pares ordenados de palabras y sus respectivos lemas.
- **Stemming:** Algoritmos que definen pares ordenados de palabras y sus respectivos temas.

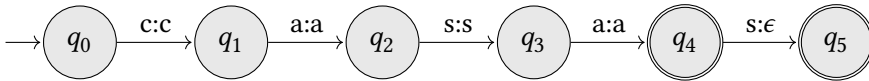


Figure 3.4: Transducer que devuelve el lema de casa o casas

#### 3.4.1.2 Lenguajes independientes de contexto

si bien los lenguajes regulares son muy útiles, existen lenguajes cuya naturaleza no puede ser capturada por un autómata de estados finitos, una expresión regular o una gramática regular, por ejemplo, aquellos lenguajes que se conocen como espejados, como el siguiente:

$$(19) \quad \{ab, aabb, aaabbb, aaaabbbb...\}$$

Para generar esta clase de lenguajes se precisa algún tipo de mecanismo que permita contar los símbolos. Una forma de hacerlo es enriquecer al autómata con una pila en la que se almacena de algún modo la información respecto de la cantidad de repeticiones del primer símbolo, para igualarla a la cantidad de

repeticiones del segundo. Este tipo de autómatas tienen el nombre de autómata de pila.

Un modo alternativo y más usual de generar estos lenguajes es mediante gramáticas independientes de contexto, esto es, gramáticas cuya única restricción es que del lado izquierdo de la regla de reescritura solo haya un símbolo no terminal.

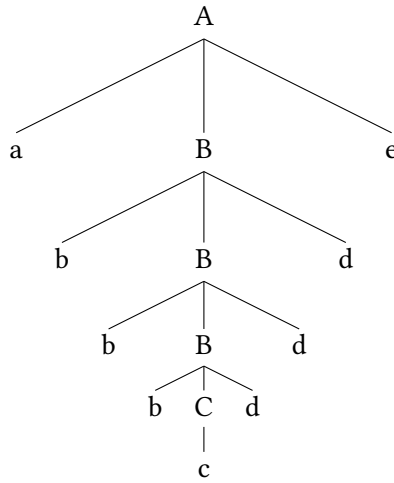
(20) Gramática Independiente de Contexto

- a.  $V_T$  = símbolos terminales = {a, b, c, d, e}
- b.  $V_N$  = símbolos no terminales = {A, B}
- c. S = el axioma inicial, incluido en  $V_N = A$
- d. R = conjunto de reglas =  $\left\{ \begin{array}{l} \text{R1. } A \rightarrow aBe \\ \text{R2. } B \rightarrow cBd \\ \text{R3. } B \rightarrow cCd \\ \text{R4. } C \rightarrow c \end{array} \right\}$

(21) Derivación para abbbcdde

- a. aBe Por R1
- b. abBde Por R2
- c. abbBdde Por R2
- d. abbbCddde Por R3
- e. abbbcdde Por R4

(22)



#### 3.4.1.3 Lenguajes sensibles al contexto

La clase de lenguajes sensibles al contexto es probablemente la menos estudiada de la jerarquía de Chomsky. Son lenguajes que pueden ser generados por autómatas linealmente acotados, esto es, máquinas de Turing que son capaces de reescribir cintas solo hasta cierto límite estipulado. Las gramáticas sensibles al contexto solo tienen como condición que el lado derecho de las reglas no borre ningún no terminal del lado izquierdo. Esta propiedad se conoce con el nombre de monotonía (*monotonicity*).

#### 3.4.1.4 Lenguajes irrestrictos

Los lenguajes irrestrictos son aquellos que se pueden generar mediante gramáticas irrestrictas, es decir, gramáticas cuyas reglas de reescritura requieren que del lado izquierdo haya al menos un no terminal. Un problema de estos lenguajes es que no son capaces de arrojar un árbol estructural.

Los lenguajes irrestrictos pueden ser generados, alternativamente, mediante máquinas de Turing.

- La máquina de Turing (MT) es el tipo de autómata más poderoso.
- Una MT es una cinta infinita con símbolos, un escaner que lee esos símbolos y un conjunto de estados con instrucciones que definen qué debe hacer en cada momento al ver un símbolo. Estas instrucciones incluyen la posibilidad de reemplazar el símbolo  $x$  que se lee por  $y$  (sean  $x$  e  $y$  iguales o

no), moverse hacia adelante o hacia atrás en la cinta y pasar del estado  $q_i$  al estado  $q_j$  (sean  $q_i$  y  $q_j$  iguales o no)

- Al correr una MT, cada aplicación de una instrucción es un paso.
- Toda función que pueda ser resuelta mediante una MT en una cantidad finita de pasos es una función computable.
- Puede optimizarse la cantidad de pasos agregando mayor cantidad de cintas con sus correspondientes escaners. Por supuesto, esto hará que cada instrucción sea más compleja. Una MT con más de una cinta se conoce como Multitape Turing Machine, y define exactamente los mismos lenguajes que las máquinas con una cinta.
- Es posible traducir cualquier máquina de Turing como una secuencia de ceros y unos.

Toda MT puede ser traducida como una computadora y toda computadora puede ser traducida como una MT. Es decir, las computadoras y las MT son equivalentes en el sentido de que definen exactamente los mismos lenguajes. Por supuesto, no son idénticas, ya que la arquitectura de cada una es diferente. Además, una máquina de Turing es un sistema abstracto que tiene todas las limitaciones formales de una computadora pero ninguna de las físicas.

#### 3.4.2 Clases de lenguajes por su capacidad de determinar pertenencia

Todo lenguaje puede traducirse como problema y viceversa de la siguiente forma:

- **Lenguaje:** Un lenguaje es un conjunto de cadenas.
- **Problema:** Un problema es la pregunta respecto de si determinada cadena pertenece a un lenguaje.

La posibilidad o no de responder un problema nos permite clasificar los lenguajes en distintos tipos:

- **Lenguajes recursivos:** Es posible decidir si un elemento pertenece a  $L$  o a  $\neg L$ . Incluye a los lenguajes regulares, los independientes de contexto, los sensibles al contexto y un subconjunto de los lenguajes irrestrictos (por ejemplo, la aritmética de Presburger).

- **Lenguajes recursivamente enumerables:** Es posible decidir si un elemento pertenece a  $L$ . Incluye a los lenguajes irrestrictos. Por ejemplo, el lenguaje universal (el conjunto de cadenas que conforman una máquina de Turing que se acepta a sí misma)
- **Lenguajes no recursivamente enumerables:** No es posible decidir si un elemento pertenece a  $L$ . Por ejemplo, el lenguaje diagonal (el conjunto de cadenas que conforman máquinas de Turing que no se aceptan a sí mismas).

#### 3.4.3 Algunos lenguajes populares

Lenguaje espejo

lenguaje copia

lenguaje universal

lenguaje diagonal

#### 3.4.4 Nociones básicas de complejidad

El principio de la **complejidad** mide la dificultad de resolver un problema computacional, medido en términos de recursos consumidos durante la computación. Normalmente se toma como referencia el espacio o el tiempo. (...) Cuanto más complejo sea el autómata permitido, tanto más complejas serán las lenguas reconocidas por él. (Moreno Sandoval 2001: 233)

En la tabla 3.6 se enumeran algunos de los tipos de costos de procesamiento más importantes en la notación de  $O$  mayúscula, que mide el tiempo máximo de procesamiento que puede llevar resolver un problema en el peor de los casos. En la figura 3.5 se comparan las respectivas curvas.

Table 3.6: Tipos de costos de procesamiento medidos en términos temporales y notación de  $O$  mayúscula

Tipos de tiempos	Notación $O$ mayúscula
Tiempo constante	$O(c)$
Tiempo lineal	$O(n)$
Tiempo polinómico	$O(n^c)$
Tiempo exponencial	$O(c^n)$
tiempo factorial	$O(n!)$



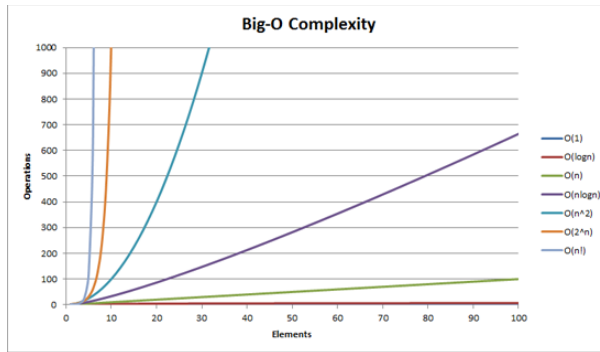


Figure 3.5: Comparación de curvas de los distintos costos de procesamiento (tomado de <https://i.stack.imgur.com/Aq09a.png>)

Se sabe que el procedimiento para parsear una cadena según una gramática tiene un costo de procesamiento según la tabla 3.7.

Table 3.7: Costos de procesamiento en términos de tiempo para el parseo de una cadena según el tipo de gramática

Clase de gramática	Tipo de tiempo
Gramáticas regulares	tiempo lineal
Gramáticas independientes de contexto	tiempo polinómico
Gramáticas sensibles al contexto	tiempo exponencial (intratable)
Gramáticas irrestrictas	indecidable

La escala de complejidad nos permite saber qué clase de problemas podemos tratar de resolver en una computadora, cuáles solo pueden ser resueltos para números pequeños de datos. En consecuencia, siempre conviene reducir los problemas a la clase de problemas más simples que podamos, aun a costa de perder efectividad.

Por ejemplo, si queremos parsear oraciones del lenguaje natural, sabemos que una gramática cualquiera que genere lenguajes independientes de contexto es insuficiente, mientras que una gramática que genere lenguajes sensibles al contexto tiene poder suficiente. No obstante, una gramática sensible al contexto es intratable, puesto que tiene un costo de procesamiento que crece exponencialmente a medida que crece la longitud de la cadena. Por esta razón, es preferible muchas veces, en todo caso, utilizar una gramática independiente de contexto y,

### 3 Jerarquía de los lenguajes formales

o bien renunciar a hacer un buen análisis de aquellas cadenas que desafían esta clase de lenguajes, o bien utilizar estrategias de posprocesamiento.

Anteriormente consideramos el siguiente lenguaje:

- $\{w \mid w \text{ es una cadena de ceros y unos y la traducción de } w \text{ en una máquina de Turing } M \text{ acepta a } w \text{ como input, es decir } w \in L(M)\}$

Este lenguaje se conoce con el nombre de **Lenguaje Universal**.

La pregunta por la pertenencia al lenguaje universal equivale en términos de lenguaje al problema de si existe un algoritmo general que pueda determinar si una Máquina de Turing puede aceptar o no determinado input. Se sabe que este problema es indecidible. Por lo tanto, cualquier intento de escribir un programa que resuelva este problema es desde ya una tarea vana.

En el campo, hay un consenso generalizado de que un algoritmo de parser descriptivamente adecuado para el lenguaje natural que pueda ser procesado eficientemente tiene que estar diseñado de modo tal de tener un costo de procesamiento polinómico.

## 3.5 Ejercicios

### 3.5.1 Autómatas

👉 Viene de página 35.

Observe el autómata de la figura 3.6 para el lenguaje  $L_7$

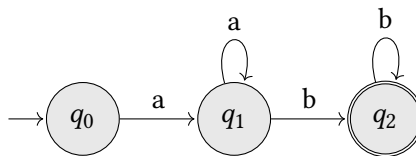


Figure 3.6: Autómata para  $L_7$

Defina por extensión el conjunto  $L_7$

(23)  $L_7 =$

Defina el autómata como una quintupla:

(24) a.  $A7 =$

b.  $A7 =$

Complete el cuadro de transición de estados en la tabla 3.8.

Table 3.8: Tabla de transición para A7


### 3.5.2 Expresiones regulares

#### 3.5.2.1 Denotación de expresiones regulares

👉 Viene de página 39.

Escriba las denotaciones para las siguientes expresiones regulares:

(25) a.  $\llbracket ab?c[da] \rrbracket =$

b.  $\llbracket [Dd]ra?\. \rrbracket =$

c.  $\llbracket \backslash d\{2\}(\backslash d\{2\}) \rrbracket =$

d.  $\llbracket \backslash w\backslash s \rrbracket =$

### 3 Jerarquía de los lenguajes formales

#### 3.5.2.2 Armado de expresiones regulares

Escriba expresiones regulares para los siguientes tipos de cadenas

- (26) a. Palabras (cadenas separadas por espacios) =
- b. Links de internet =
- c. Oraciones (cadenas que terminen en un punto) =

#### 3.5.3 Gramáticas regulares

👉 Viene de la página 41.

Suponga la siguiente gramática:

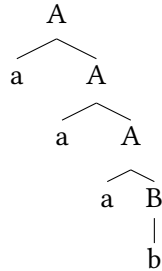
(27) **Gramática 1**

- a.  $V_T$  = símbolos terminales = {a, b, c}
- b.  $V_N$  = símbolos no terminales = {A, B}
- c. S = el axioma inicial, incluido en  $V_N = A$
- d. R = conjunto de reglas = 
$$\left\{ \begin{array}{l} \text{R1. } A \rightarrow aA \\ \text{R2. } A \rightarrow aB \\ \text{R3. } B \rightarrow b \\ \text{R4. } B \rightarrow c \end{array} \right\}$$

(28) **Derivación para aaab**

- a. aA por R1
- b. aaA por R1
- c. aaaB por R2
- d. aaab por R3

(29) Estructura para aaab



Escriba la expresión regular equivalente a este lenguaje:

### 3.5.4 Tipos de reglas gramaticales

¿Cuál es la clase mínima de gramáticas a las que pueden pertenecer las siguientes reglas?

(30) a.  $D \rightarrow e$ b.  $a B c \rightarrow d c$ c.  $a B c \rightarrow a d c$ d.  $A \rightarrow B C$ e.  $A \rightarrow c B$ f.  $A b \rightarrow B$ 

### 3.5.5 Repaso

Determine si las afirmaciones son verdaderas o falsas y justifique en ambos casos.

### *3 Jerarquía de los lenguajes formales*

- (31) a. Una gramática genera débilmente secuencias de símbolos.
- b. Todo lenguaje  $L$  que se pueda construir sobre el alfabeto  $\Sigma$  está incluido en el conjunto  $\Sigma^*$
- c. Un autómata de pila puede reconocer un lenguaje regular.
- d. El problema del reconocimiento universal es el problema de reconocer si una cadena pertenece al lenguaje universal.
- e. Una oración agramatical es una oración que pertenece al complemento del lenguaje en cuestión.
- f. Los autómatas generan fuertemente estructuras.
- g. El complemento de un lenguaje recursivo también es necesariamente un lenguaje recursivo

## 3.6 Recapitulación

### 3.7 Lecturas recomendadas

- J. Daniel Quesada. 1974. *La lingüística generativo transformacional: supuestos e implicaciones*. Madrid: Alianza. “Capítulo 3: Las gramáticas generativas como sistemas formales”, pp. 29–36.
- Shuly Wintner. 2010. Formal language theory. In Alexander Clark et al. (eds.), *The handbook of computational linguistics and natural language processing*, 11–42. Massachusetts: Willey Blackwell
- María Teresa Solias Aris. 2015. *Métodos formales en lingüística*. Síntesis. Capítulo 2. “Las gramáticas formales”, pp. 43–94.
- Brian Roark & Richard Sproat. 2007. *Computational approaches to morphology and syntax*. Oxford: Oxford University Press
- Barbara Partee et al. 2012. *Mathematical methods in linguistics*. Dordrecht: Kluwer Academics





## **Part II**

# **Las gramáticas basadas en constituyentes**



## 4 Gramáticas independientes de contexto

### 4.1 Introducción

En la primera parte de este libro hemos presentado las principales nociones de teoría de los lenguajes: alfabeto, cadena, lenguaje, autómata y gramática formal, entre otras. En esta segunda parte nos vamos a concentrar en los tipos de gramáticas basadas en la noción de constituyente. Probablemente, este es el tipo de gramática más prototípico, ya que la noción de constituyente está muy asentada en la tradición lingüística, al menos desde [Bloomfield 1933](#). En este primer capítulo nos vamos a centrar en lo que es posiblemente uno de los tipos de gramáticas más estudiados: las gramáticas independientes de contexto. En los capítulos anteriores hemos visto que no todos los lenguajes, entendidos como conjuntos de cadenas, están sujetos a las mismas restricciones ni tienen la misma complejidad. Por ejemplo, hay un tipo de producción que los lenguajes regulares no pueden describir. Veamos un ejemplo sacado de [Wintner 2010](#).

Consideremos una lengua sobre el alfabeto  $\Sigma = \{a,b\}$  cuyos miembros son cadenas que consisten en un número  $n$  de  $a$  seguido del mismo número de  $b$ s

Por ejemplo, serán miembros de esta lengua cadenas como las de (1):

(1)  $\{aabb, aaabbb, ab...\}$

Formalmente esta lengua se puede anotar como... (tildar la opción correspondiente)

- 1.  $L = \{a^n \cdot b^n | n > 0\}$
- 2.  $L = \{a^n \cdot b^m | n > 0\}$
- 3.  $L = \{a^* b^*\}$

Asumamos que  $L$  es un lenguaje regular y que, por lo tanto, existe un autómata de estado finito cuya lengua es  $L$ .

## 4 Gramáticas independientes de contexto

Ahora consideremos una lengua  $L_i = \{ a^i \mid i > 0 \}$ .

Siguiendo lo que sabemos de  $L$ , podemos decir que todas las cadenas de  $L_i$  son prefijos de alguna cadena de  $L$ .

Entonces, para todo string en  $L_i$  tiene que haber un camino posible en el autómata de  $L$  que lleve del estado inicial  $q_0$  al siguiente estado.

Si bien hay infinitas cadenas en  $L_i$ , hay finitos estados en el autómata, por definición. Por lo tanto tenemos al menos dos cadenas en  $L_i$  que nos van a llevar del mismo estado inicial al mismo estado de llegada  $q_f$ . Llamemos a estas cadenas  $a^j$  y  $a^k$ , donde  $j \neq k$  (pongámosle,  $j=3$ ,  $k=4$ , es decir “aaa” y “aaaa” respectivamente). El problema es que, si vemos el autómata resultante para  $L$ , partiendo de esta necesidad, podemos comprobar que ese autómata ya no es equivalente a  $L$ . ¿Por qué?

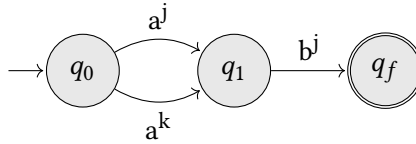


Figure 4.1: Autómata para  $L$

Porque este autómata reconoce las cadenas  $a^j b^j$ , que pertenece a  $L$ , pero también acepta  $a^k b^j$ , que no pertenece a  $L$ .

Esta es una forma de demostrar que hay lenguajes que no son regulares y que hay una clase más compleja de lenguajes, lo que nos da pie a introducir los lenguajes independientes de contexto y las gramáticas independientes de contexto que los generan.

Vamos a revisar también los axiomas propios de estas gramáticas, que son los axiomas de dominancia y de precedencia.

Y por último, antes de verlos en acción, vamos a presentar los “parsers” que son programas que nos van a permitir aplicar las gramáticas independientes de contexto para producir el análisis sintáctico (árbol) de una oración.

## 4.2 Gramáticas independientes de contexto

### 4.2.1 Definición

Los lenguajes independientes de contexto son de tipo 2. Son lenguajes con mayor poder expresivo que los regulares lo que también implica mayor costo de procesamiento. Esto lo vimos la clase pasada cuando vimos la notación  $O$  y vimos que estos lenguajes se pueden procesar en tiempo polinómico.

Si bien este costo de procesamiento es menor al de los lenguajes dependientes de contexto y por eso estos lenguajes son “atractivos” para el procesamiento del lenguaje, se suele considerar a los lenguajes independientes de contexto como insuficientes para representar a las lenguas naturales.

Como ya vimos, una forma de describir los lenguajes formales es por medio de una gramática y una gramática formal está constituida por cuatro elementos.

$$(2) \quad G = \langle V_T, V_N, S, R \rangle$$

Para una gramática independiente de contexto que describa una lengua natural, podemos representar esa cuádrupla del siguiente modo:

- (3) a.  $V_T$  = un vocabulario de símbolos terminales, que nos permitirá tener almacenadas y definidas las entradas léxicas.
- b.  $V_N$  = un conjunto de símbolos no terminales, como SN, SV, etc., que nos permitirán agrupar las categorías en constituyentes y sintagmas hasta llegar a la categoría raíz.
- c.  $S$  = un símbolo raíz de la categoría gramatical más compleja que se considere, normalmente oración (o  $S$  por *sentence*).
- d.  $R$  = un conjunto de reglas de derivación.

Cuando usemos estas gramáticas con los programas parser, las vamos a describir directamente por medio de sus reglas. Las reglas nos van a describir la estructura interna de los constituyentes de una gramática.

La notación de estas reglas consiste en una secuencia compuesta por un símbolo no terminal y el signo  $\rightarrow$  seguido por una secuencia de símbolos terminales o no terminales

¿Cómo serán las reglas para describir a  $L$ ?

### 4.2.2 Derivación

El símbolo  $\rightarrow$  establece una relación de derivación entre dos formas pertenecientes a la gramática. Y es la relación de derivación lo que permitirá definir el lenguaje que describe una gramática.

Veamos las reglas de una gramática simple.

- (4) Derivaciones de un lenguaje  $L_1$

#### 4 Gramáticas independientes de contexto

- a.  $O \rightarrow SN\ SV$
- b.  $SN \rightarrow Det\ N$
- c.  $SV \rightarrow V$
- d.  $Det \rightarrow el\ | la\ | los\ | las$
- e.  $N \rightarrow perro\ | vaca\ | pollitos\ | gaviotas$
- f.  $V \rightarrow ladra\ | muge\ | pían\ | graznan$

¿Qué oraciones serán generadas por esta gramática?

##### 4.2.3 Árboles

Los árboles de derivación presentan una ayuda para visualizar derivaciones de estas gramáticas y también los veremos como *output* de los parsers.

Los árboles son grafos y pueden ser descriptos de un modo formal dentro de la teoría matemática de grafos, que estudia las propiedades de estos objetos que representan relaciones binarias dentro de un conjunto. Un uso habitual es, por ejemplo, el estudio de las relaciones de individuos o cuentas en redes sociales.

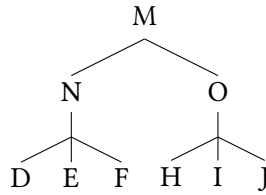
Los grafos se describen por medio de dos conjuntos: el conjunto de vértices (nodos del árbol) y conjunto de aristas (ramas del árbol), que se definen como el par de nodos que cada arista conecta. Como veremos más adelante, hay un orden jerárquico en los árboles de derivación que organiza los pares que denominan las aristas, por lo cual, estos son pares ordenados.

El árbol tiene siempre un nodo raíz, único, desde el que parten las derivaciones y que para nosotros estará etiquetado con el símbolo  $O$ . Los nodos finales del árbol suelen aparecer etiquetados con los símbolos terminales, o entradas léxicas ( $V_T$ ), de la gramática. En los nodos intermedios, vamos a encontrar los símbolos intermedios no terminales ( $V_N$ ).

Las ramas del árbol representan las relaciones entre los nodos.

Veamos un ejemplo:

(5)



En este grafo tenemos el conjunto de vértices enunciado en (6):

(6)  $\{M, N, O, D, E, F, H, I, J\}$ .

Y si bien no hemos usado flechas para dibujar debemos asumir que las ramas son flechas que viajan en una sola dirección: hacia abajo. Por esto el conjunto de aristas está formado por pares ordenados. El conjunto de pares ordenados para el árbol 5 se enuncia en (7):

(7)  $\{ \langle M, N \rangle, \langle M, O \rangle, \langle N, D \rangle, \langle N, E \rangle, \langle N, F \rangle, \langle O, H \rangle, \langle O, I \rangle, \langle O, J \rangle \}$ .

Esto es así porque los árboles de derivación son grafos acíclicos dirigidos. Es decir, que las aristas tienen una dirección y, a su vez, no es posible formar un ciclo entre nodos.

¿Cuál es el nodo raíz?

¿Cuáles son los nodos intermedios?

¿Cuáles son los nodos terminales?

¿Cómo se denomina la arista que va de M a N?

#### 4.2.4 Axiomas de dominancia y precedencia

Por lo visto hasta acá, podemos asumir que entre los nodos se establecen relaciones de arriba hacia abajo y, de un modo intuitivo, que también las relaciones se establecen de izquierda a derecha (ya que conocemos el orden de la secuencia de símbolos terminales). Estas relaciones se conocen como dominancia (D) y precedencia (P) sobre un conjunto N de nodos.

#### 4.2.4.1 Dominancia

Informalmente, podemos entender dominancia como la relación que un nodo que se encuentra más arriba en el árbol tiene con un nodo de más abajo. En nuestro árbol 5, por ejemplo, M domina a N y a O. Y M domina también a D, E, F, H, I y J. En este sentido, la dominancia es también una relación de contención: M contiene a todos los demás nodos del árbol. En una oración, diríamos que la categoría superior, O, domina a todos los sintagmas de la oración.

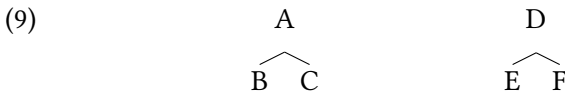
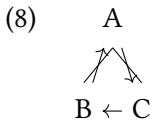
Podemos, entonces, redefinir los símbolos de nuestra gramática:

- El nodo raíz es el nodo que domina a todos y que no es dominado por ningún otro (excepto sí mismo).
- Los nodos intermedios (no terminales) son los que dominan a otros (y a sí mismos).
- Los nodos terminales serán aquellos que no dominan a ningún nodo (excepto a sí mismos).

Matemáticamente, podemos formular los axiomas que describen formalmente la relación de dominancia (D).

- A1. D es reflexiva:  $(\forall x \in N) [x \triangleleft^* x]$ .

Esto lo hemos visto al decir que los nodos, de todo tipo, se dominan a sí mismos y va a ser importante para excluir árboles de múltiples raíces y árboles donde la raíz sea dominada por otro nodo (Esto también implica algo que ya sabemos: los árboles derivacionales son acíclicos):



Para excluir árboles como el de 8, 9 necesitamos otro axioma:

- A2. Condición de raíz única:  $(\exists x \forall y \in N) [x \triangleleft^* y]$ .



Este axioma excluye a los árboles con más de una raíz e indica que todo nodo debe tener un único nodo que lo domine. Para el nodo raíz esto es posible únicamente al ser D reflexiva.

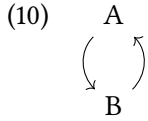
¿Se les ocurre una frase del español que no cumpla este axioma?

El siguiente axioma nos dice que:

- A3. D es transitivo:  $(\forall xyz \in N) [((x \triangleleft^* y) \& (y \triangleleft^* z)) \rightarrow (x \triangleleft^* z)]$ .

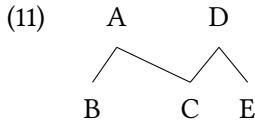
Esto también lo hemos visto al decir que, en nuestro árbol, M no sólo dominaba a N y O, sino también a todos los demás nodos inferiores.

- A4. D es antisimétrico:  $(\forall xy \in N) [((x \triangleleft^* y) \& (y \triangleleft^* x)) \rightarrow (x = y)]$ .



Este axioma nos indica también la propiedad dirigida y acíclica de los grafos que estamos analizando, ya que, si un nodo x domina a un nodo y, y dicho nodo y domina a su vez al nodo x, eso significa sí o sí que x e y son el mismo nodo ( $x = y$ ), porque si no lo fueran, se rompería el axioma A4..

- A5. Madre única:  $(\forall xyz \in N) [((x \triangleleft^* z) \& (y \triangleleft^* z)) \rightarrow ((x \triangleleft^* y) \vee (y \triangleleft^* x))]$ .



A5. nos permitirá excluir árboles como el de 11, ya que los nodos no pueden tener más de una madre o, en otros términos, un único nodo que lo *domina inmediatamente*, informalmente conocido como “madre”, y del que, dicho nodo, es inmediatamente dominado (hijo). También se denominan nodos hermanos aquellos nodos inmediatamente dominados por el mismo nodo madre. La relación de dominancia inmediata es útil para referirse a relaciones locales entre nodos y es la que vemos reflejada en los pares ordenados de vértices que definen en

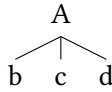
#### 4 Gramáticas independientes de contexto

teoría de grafos el conjunto de aristas. Simbolizaremos la relación de *dominancia inmediata* como  $\triangleleft^+$ .

Otra relación importante para nosotros es la de *dominancia exhaustiva* que es la que se da entre un conjunto de nodos terminales y un nodo que los domina a todos y que no domina a más nodos que los que forman parte de ese conjunto.

En el ejemplo 12, A *domina exhaustivamente* al conjunto de nodos {b, c, d} y no hay ningún otro nodo, llamémoslo x, que esté dominado por A y no es parte del conjunto.

(12)



Entonces, basandonos en los axiomas de dominancia y la relación de dominancia exhaustiva podemos definir qué es un constituyente:

Un constituyente es un conjunto de nodos *dominados exhaustivamente* por un único nodo.

##### 4.2.4.2 Precedencia

Informalmente, la precedencia es lo que “se lee antes”. En la oración anterior, “Informalmente” precede a “la”, “la” a “precedencia” y así sucesivamente. Formalmente, la precedencia tiene implicancias importantes en la concepción de las gramáticas y la vamos a anotar con el símbolo  $\prec$ .

Digamos primero que en la precedencia también se refleja la relación de “árbol familiar” entre nodos. En este caso, entre nodos hermanos:

- **Nodos hermanos:** Un nodo precede a su nodo hermano si los dos nodos están dominados inmediatamente por la misma madre y el nodo que precede emerge de una rama colocada a la izquierda de su nodo hermano.

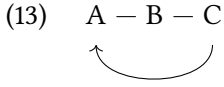
También vamos a definir una relación de precedencia inmediata entre nodos:

- **Precedencia inmediata:** Un nodo precede inmediatamente a otro si el primero está inmediatamente a la izquierda del nodo al que precede.

Con estas definiciones en mente, podemos definir las propiedades de la precedencia (P) de forma axiomática como lo hicimos con la dominancia (D):

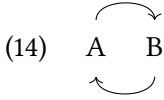
- **A6.** P es transitiva:  $(\forall xyz \in N) [((x \prec y) \& (y \prec z)) \rightarrow (x \prec z)]$ .

Esto excluye árboles como:



Si lo pensamos en la cadena sonora, sería como pronunciar A antes de B, B antes de C y C antes de A... Tampoco podríamos pronunciar dos cosas al mismo tiempo, por eso:

- A7. P es asimétrica:  $(\forall xy \in N)[((x < y) \rightarrow \neg(y < x))]$



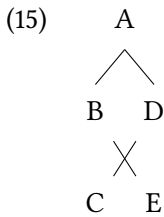
De este axioma se sigue que la precedencia no es reflexiva:

- T1. P es irreflexiva:  $(\forall x \in N) [\neg(x < x)]$

A su vez, entre P y D hay una condición de exclusividad, es decir que si un nodo domina al otro, no puede precederlo y viceversa. Con esta condición, P y D organizan el total de relaciones posibles en nuestro conjunto de nodos y aristas.

- A8. Condición de exclusividad:  $(\forall xyz \in N) [((x < y) \vee (y < x)) \leftrightarrow \neg((x \triangleleft^* y) \vee (y \triangleleft^* x))]$ .

Por último, la precedencia cumple con otra condición muy importante para nosotros, que es la condición de “no enredamiento”, es decir, que no se puede dar algo como el siguiente árbol:



- A9. Non-tangling condition:  $(\forall wxyz \in N) [((w <_s x) \& (w \triangleleft^* y) \& (x \triangleleft^* z)) \rightarrow (y < z)]$ .

Este axioma refleja la asunción de Chomsky de que no existen constituyentes discontinuos, o, en nuestro árbol 15, que las ramas no pueden cruzarse.

¿Se les ocurre una frase del español que no cumpla este axioma?

### 4.2.5 Chomsky Normal Form

Hasta aquí asumimos que las reglas de derivación podían tomar la forma de un símbolo no terminal a la izquierda del símbolo de derivación  $\rightarrow$ , y de cualquier secuencia de símbolos no terminales o terminales a la derecha.

Esta forma se puede normalizar sin afectar el poder generativo de la gramática y a nosotros nos va a ser útil esta normalización para alguna de las aplicaciones de parser.

La forma normalizada que vamos a ver se conoce como Chomsky Normal Form y restringe la definición de las reglas a dos formas: del lado derecho de la derivación debe haber un solo símbolo terminal o exactamente dos símbolos no terminales:

Si volvemos a mirar algunos ejemplos que fuimos usando hasta ahora, ¿cuál respeta la CNF y cuál no?

- 1.  $SN \rightarrow \text{Det } N$
- 2.  $S \rightarrow aSb$
- 3.  $V \rightarrow \text{ladra} \mid \text{muge} \mid \text{pían} \mid \text{graznan}$

### 4.2.6 Algunas limitaciones de las CFG

Durante este recorrido por la definición de las gramáticas independientes de contexto ya hemos visto algunas imitaciones respecto a su poder descriptivo de una lengua natural: oraciones con más de una raíz y constituyentes cruzados. Veamos alguna de esas limitaciones formalmente.

#### 4.2.6.1 Constituyentes discontinuos y dependencias cruzadas

El lenguaje que consiste de una secuencia de símbolos y su imagen espejada se conoce con el nombre de **lenguaje espejado o palindrómico**:

(16)  $\{aa, bb, abba, aabbaa, bbbb, baab, aabbbbbaa, ababbaba...\}$

Los lenguajes espejados son lenguajes independientes de contexto.

En el lenguaje natural, se puede encontrar una muestra de lenguajes espejados en el fenómeno del *central-embedding*.

- (17) Hans Peter Marie schwimmen lassen sah  
Hans Peter Marie nadar            hacer   vio  
'Hans vio a Peter hacer nadar a María'.

Un tipo de lenguaje muy similar a los lenguajes espejados es el llamado **lenguaje copia** (*copy language*). Partee *et al* (2012: 534) lo define del siguiente modo:

- (18)  $\{xx \mid x \in \{a,b\}^+\}$

Es decir, el lenguaje copia es un conjunto de cadenas formadas por la concatenación de dos subcadenas idénticas.

- (19)  $\{aabaaaba, aaaa, abab, abaaba, aaabaaab, bbabba...\}$

Sin embargo, a pesar de su semejanza con los lenguajes espejados, el lenguaje copia se distingue porque implica dependencias cruzadas en lugar de espejadas.

Para dar cuenta de esto, una gramática independiente de contexto es insuficiente y se precisa, en su lugar, una gramática sensible al contexto.

Ahora bien, ¿existen fenómenos lingüísticos que respondan a este tipo de esquema? Sí. Un ejemplo son las dependencias cruzadas que se dan por causa del llamado *salto del afijo*:

Un ejemplo más radical de dependencias cruzadas son las llamadas dependencias cruzadas del holandés:

¿Existe un fenómeno similar a este en español? La clase pasada vimos uno que según algunos autores se ajusta a esta descripción: las oraciones con *respectivamente*.

- (20) a. Catalina y Martín tomaron respectivamente un Campari y un Fernet.  
b. Catalina, Martín y Federico tomaron respectivamente un Campari, un Fernet y una cerveza.  
c. Catalina, Martín, Federico y Victoria tomaron respectivamente un Campari, un Fernet, una cerveza y un aperitivo.

#### 4.2.6.2 Proliferación de reglas

Por último, en la ejercitación van a poder apreciar cómo el manejo de un fenómeno como la concordancia multiplica de manera explosiva las reglas de estas

#### 4 Gramáticas independientes de contexto

gramáticas. Si bien esto no representa estrictamente una limitación, como lo veremos en versiones más complejas de las CFG en las próximas clases, sí nos ayuda a entender qué problemas se presentan al usarlas.

Veámoslo en un ejemplo breve:

- $O \rightarrow SN\ SV$
- $SN \rightarrow Det\ N$
- $Det \rightarrow el\ |\ la$
- $N \rightarrow casa\ |\ cohete$
- $SV \rightarrow V$
- $V \rightarrow vuela$

Esta gramática produce secuencias gramaticales como “el cohete vuela” y “la casa vuela”. Pero también produce secuencias agramaticales como “la cohete vuela” y “el casa vuela”.

Para que esto no suceda, deberíamos agregar símbolos no terminales que indiquen la diferencia de género:

- $O \rightarrow SN_{fem}\ SV$
- $O \rightarrow SN_{masc}\ SV$
- $SN_{fem} \rightarrow Det_{fem}\ N_{fem}$
- $SN_{masc} \rightarrow Det_{masc}\ N_{masc}$
- $Det_{fem} \rightarrow la$
- $Det_{mas} \rightarrow el$
- $N_{fem} \rightarrow casa$
- $N_{masc} \rightarrow cohete$
- $SV \rightarrow V$
- $V \rightarrow vuela$

Esto implica no solamente que las reglas se multiplican con cada aspecto de estos fenómenos, sino también que entendemos estos fenómenos como axiomas dados y no podemos dar cuenta de ninguna otra explicación sobre ellos.

### 4.2.7 Ejercitación

1. Dibuje todos los árboles posibles para las siguientes oraciones:

- Fernando miró al perro con rabia.
- Macarena saludaba a la vecina con la escoba.
- Pablo estornudó fuertemente.

2. Escriba las reglas derivativas de una gramática que genere las siguientes oraciones (y no otras):

- El león está cansado.
- La chita está cansada.
- Los leones están cansados.
- Las chitas están cansadas.

(21) **Pequeña ayuda para 2:**

$O \rightarrow \text{SNplfem SVplfem}$

$O \rightarrow \text{SNsgfem SVsgfem}$

$\dots \rightarrow \dots$

### 4.2.8 Parsers

Un parser es un programa que toma un “*input*”, generalmente una cadena de caracteres (*string*), y devuelve una estructura de datos. La acción que el parser realiza la vamos a llamar, entre nosotros, “parsear” (*parsing*). Parsear es el proceso de analizar una cadena de símbolos, que pueden ser símbolos pertenecientes al lenguaje natural o a un lenguaje formal, según las reglas de una gramática formal,

para llegar a una estructura de datos. Un parser es lo que, por ejemplo, tomará el código de python que veremos en clase y devolverá un “parse tree” que permite que el código “humano” de python se convierta en código ejecutable por una máquina. De este modo podemos ver que en computación, el uso de programas de parseo excede su uso en lenguas naturales, pero el funcionamiento es el mismo en cierta medida.

### 4.2.9 Parsers sintáctico

Dada una gramática y una oración, el proceso de parseo es el proceso de asignar a la oración la estructura sintáctica correspondiente, de acuerdo a la gramática, y así determinar si una oración es gramatical o no. Es decir, en nuestro contexto, parsear es analizar sintácticamente una oración partiendo de una gramática y devolver el árbol correspondiente si esta oración pertenece efectivamente a la gramática (y si no, también... en muchos casos que vamos a ir viendo).

Un “parser” es, por lo tanto, un analizador sintáctico automatizado que recibe una gramática y una oración como “*input*” y aplica en una serie de pasos las reglas de la gramática sobre la oración hasta conseguir como “*output*” el análisis sintáctico subyacente a la cadena.

Si bien “*input*” y “*output*” suelen ser los mismos, no todos los parsers son iguales. Según cómo apliquen las reglas de la gramática, los hay de distinto tipo:

- **Top-Down:** El parser parte del símbolo terminal de la gramática y desarrolla cada una de las reglas posibles de derivación hasta encontrar la que coincide con el texto de la oración.
- **Bottom-Up:** El parser parte de la cadena de palabras de la oración y les asigna un las posibles categorías no terminales que irá combinando según las reglas hasta llegar a la categoría raíz.
- **Dynammic Programming:** El parser guarda resultados intermedios de la aplicación de reglas que podrá ir a buscar para completar la tarea de análisis de un modo efectivo.
- **Probabilistic Parsers:** Se basan en gramáticas que asignan peso (o probabilidad) a las estructuras para producir el análisis más probable.

A su vez, para estas variedades, encontraremos diferentes programas que los implementan de modos levemente distintos.



## **4.3 Recapitulación**

## **4.4 Lecturas recomendadas**



# 5 Gramática independiente de contexto generalizada (GPSG)

## 5.1 Introducción

Generalized Phrase Structure Grammar (GPSG) es una teoría gramatical desarrollada por un conjunto de investigadores que pretendía desarrollar un marco teórico con las siguientes características:

- Que fuese descriptivamente adecuado
- que fuese una gramática explícita que pudiera ser implementada en computadora de modo eficiente

Para entender las motivaciones que hicieron que GPSG adoptara la forma que adoptó, hay que tener en cuenta algunas cuestiones del contexto histórico:

- Luego de fracasos en el intento de construir algoritmos de parsing para gramáticas transformacionales y del trabajo de [Peters & Ritchie 1973](#) que demostraba que el modelo de [Chomsky 1965](#) era indecidible, existía bastante escepticismo respecto de que una gramática en los lineamientos de la gramática generativa transformacional de aquellos años pudiera implementarse computacionalmente.
- Se había demostrado recientemente que las gramáticas independientes de contexto permiten parsear oraciones en tiempo polinómico, más precisamente, en  $O(n^3)$  (ver [Valiant 1975](#) [Graham & Harrison 1976](#)). Incluso, [Earley 1970](#) había propuesto un algoritmo que permitía parsear algunos lenguajes independientes de contexto en tiempo polinómico y otros en tiempo lineal.
- [Chomsky 1957](#) había demostrado que las lenguas naturales no son lenguajes regulares, pero no había demostrado que no fueran lenguajes independientes de contexto. Su motivación para postular una gramática transformacional era conceptual. Para esta época no se había demostrado tajantemente que las lenguas naturales no fueran lenguajes independientes de contexto.

## 5 Gramática independiente de contexto generalizada (GPSG)

Las conclusiones a las que arriban los propulsores de GPSG a partir de esto son las siguientes:

- Hay que armar una gramática que respete las convenciones de las gramáticas independientes de contexto, para de ese modo conservar la decidibilidad en  $O(n^3)$ .
- Hay que enriquecer esa gramática con algo que, sin aumentar su capacidad generativa, permita solucionar los problemas conceptuales señalados en [Chomsky 1957](#).

### 5.1.1 Convenciones de las Gramáticas independientes de contexto

Como vimos en clases anteriores, las gramáticas se definen como cuádruplas de la forma  $\langle NT, ST, SI, RP \rangle^1$ , en la que los símbolos NT, T, O y P deben entenderse del siguiente modo:

- **NT = Símbolos no terminales o símbolos auxiliares:** Los símbolos
- **T = Símbolos terminales:** Se los suele escribir en minúscula para distinguirlos de los símbolos no terminales.
- **O = Símbolo inicial:** Un símbolo no terminal que encabeza la estructura de la cadena a analizar tal que  $O \in NT$ .
- **P = Reglas de Producción:**

Ver ejercicio 5.3.1 Hasta acá, la definición de arriba aplica no solo a las gramáticas independientes de contexto, sino también a todo tipo de gramática formal. La característica definitoria de las gramáticas independientes de contexto está dada fundamentalmente por el tipo de reglas de reescritura que aceptan. Estas reglas pueden reescribir solo un símbolo no terminal del lado izquierdo y no poseen restricciones respecto de lo que pueda aparecer del lado derecho de la regla.

Los autores ligados a GPSG interpretan entonces que las reglas de su gramática tienen que respetar esta restricción básica de que del lado izquierdo solo puede aparecer un no terminal.

---

<sup>1</sup>Según el autor, puede variar el orden de los elementos de la cuádrupla o la definición de los elementos.

## 5.1.2 Problemas conceptuales de las Gramáticas independientes de contexto

### 5.1.2.1 Problemas de linealidad

Como vimos, las reglas independientes de contexto permiten asignar una estructura a una cadena. La estructura se puede representar como un grafo acíclico dirigido (generalmente orientado de arriba hacia abajo) en el que cada bifurcación se corresponde a una regla de reescritura en la que el elemento a la izquierda de la regla es representado por el nodo superior y los elementos a la derecha por los nodos inferiores. Estos árboles se construyen de tal modo que respetan el orden lineal de cada lado derecho de la regla y no permite que los arcos se crucen.

Ver  
ejercicio  
5.3.2

Ahora bien, existen al menos dos argumentos para preferir que las reglas independientes de contexto no codifiquen relaciones de precedencia lineal

- Existen lenguas que permiten órdenes alternativos de palabras, y eso implicaría la necesidad de escribir distintas reglas independientes de contexto para las mismas relaciones jerárquicas.
- Dos lenguas que se diferencien en si colocan el núcleo a la derecha o a la izquierda precisarían reglas independientes de contexto diferentes, aunque las relaciones jerárquicas entre los constituyentes fueran las mismas en las dos.

Ver  
ejercicio  
5.3.3

### 5.1.2.2 Problemas de parentesco

Otro problema de las gramáticas independientes de contexto son los problemas de parentesco (*relatedness*)

Un ejemplo típico es el de la pasiva:

- (1) a. Borges escribió *Ficciones*.  
b. *Ficciones* fue escrito por Borges

Podemos generar una gramática que genere las dos oraciones:

- (2) a.  $O \rightarrow \text{SNanim SVact}$   
b.  $O \rightarrow \text{SNinanim SVpas}$   
c.  $\text{SNanim} \rightarrow \text{Borges}$   
d.  $\text{SNinanim} \rightarrow \text{Ficciones}$   
e.  $\text{SVact} \rightarrow \text{V SNinanim}$

## 5 Gramática independiente de contexto generalizada (GPSG)

- f.  $SV_{pas} \rightarrow PerPas SP$
- g.  $V \rightarrow \text{escribió}$
- h.  $PerPas \rightarrow \text{fue Part}$
- i.  $Part \rightarrow \text{escrito}$
- j.  $SP \rightarrow P SN_{anim}$
- k.  $P \rightarrow \text{por}$

Sin embargo, este tipo de formalización pierde la intuición que tienen los hablantes de que, en líneas generales, la oración (1b) se corresponde con la versión pasiva de la oración (1a). Existe una relación de parentesco (*relatedness*) entre activas y pasivas que una gramática independiente de contexto es incapaz de reflejar. Por el contrario, una gramática transformacional como la de Chomsky 1957, explica esa relación de parentesco a partir de la apelación a la estructura profunda.

### 5.2 GPSG: el formalismo

GPSG es una de las primeras gramáticas que complementa la enunciación de reglas con la enunciación de principios generales que todas las reglas deben cumplir, reduciendo así el sistema axiomático, aunque incrementándole la ontología de reglas. El sistema pasa de tener solamente reglas de reescritura a tener en su lugar el siguiente tipo de reglas:

- reglas ID
- reglas LP
- metarreglas
- reglas léxicas
- postulados de significado
- rasgos y sus convenciones asociadas:
  - restricciones de coaparición de rasgos
  - especificación de rasgos por *default*.

### 5.2.1 Las reglas ID/LP

Para evitar el problema relacionado con la linealidad, en GPSG se separan las reglas que introducen las relaciones jerárquicas de las reglas que introducen las relaciones de precedencia lineal. De este modo, existen dos grandes tipos de reglas:

- Reglas ID (Reglas de dominancia inmediata o *ID rules*): establecen exclusivamente las relaciones de jerarquía.
- Reglas LP (Reglas de precedencia lineal o *LP rules*): restringen los órdenes posibles.

Las reglas ID se anotan separando los símbolos del lado derecho con comas.

#### (3) Reglas de Dominancia Inmediata:

- $S \rightarrow NP, VP$
- $NP \rightarrow N$
- $VP \rightarrow V, NP_{[a]}$
- $NP_{[a]} \rightarrow a, SN$
- $V \rightarrow AUX, VLEX$

Estas comas pretenden indicar que los elementos no se suceden el uno al otro, sino que simplemente son hijos de la categoría a la izquierda de la flecha.

Las reglas LP codifican la linealidad con el símbolo  $\prec$  que indica que lo que está a su izquierda debe aparecer a la izquierda de lo que aparezca a su derecha.

#### (4) Reglas de precedencia lineal

- $NP \prec VP$
- $AUX \prec VLEX$
- $a \prec NP$

¿Qué órdenes posibles da a lugar la combinación de las reglas ID de (3) y las reglas LP de (4)?

### 5.2.2 El uso de rasgos y sus convenciones asociadas

En GPSG, los nodos no terminales ya no son símbolos primitivos, sino que son conjuntos formados por rasgos, es decir, pares de <atributo, valor>. Estos rasgos y sus posibles valores asociados incluyen los siguientes:

- $CAT \times \{N, V, P, A...\}^2$
- $CASE \times \{ACC, NOM\}$
- $PER \times \{1, 2, 3\}$
- $PLU \times \{+, -\}$
- $BAR \times \{0, 1, 2\}^3$
- $COMP \times \{que, si\}$

Así, el nodo correspondiente a un nombre como *vaca*, en un árbol de derivación clásico se representa como en (5a). En GPSG se representa como en (5b)

- (5) a. Nodo correspondiente a *vaca* en una gramática independiente de contexto típica

N  
|  
vaca

---

<sup>2</sup>En realidad, en GPSG las categorías se obtienen mediante la combinación de los valores positivo y negativo para los rasgos N y V.

<sup>3</sup>Müller 2016 utiliza 1, 2 y 3 como valores, puesto que sigue en este punto la notación de Uszkoreit 1987, que es una de las figuras principales de la implementación de GPSG para el alemán. La notación originaria, presente en Gazdar et al. 1985, es 0, 1 y 2.



b. Nodo correspondiente a *vaca* en una gramática al estilo de GPSG

$$\begin{array}{c} \{ \langle \text{CAT}, \text{N} \rangle, \langle \text{PLU}, - \rangle, \langle \text{GEN}, \text{fem} \rangle, \langle \text{BAR}, 0 \rangle \dots \} \\ | \\ \text{vaca} \end{array}$$

Los valores para el rasgo BAR indican el nivel de X con barra. Se suelen anotar simplifcadamente como un superíndice, dando lugar a la equivalencia en (6):

$$(6) \quad \text{VP} \rightarrow \text{V}, \text{NP} = \text{V}^2 \rightarrow \text{V}, \text{N}^2$$

- Las proyecciones mínimas (y solo ellas) contienen un rasgo de subcategorización al que llaman SUBCAT. El valor de SUBCAT es un número.
- Las proyecciones intermedias ( $\text{X}^1$ ) y máximas ( $\text{X}^2$ ) tienen el rasgo de SUBCAT indefinido. Esto se escribe como  $\sim[\text{SUBCAT}]$
- El rasgo SUBCAT, en una notación simplificada, se marca entre corchetes como un subíndice.
- En una regla de la forma “ $\text{X} \rightarrow \dots \text{H} \dots$ ”, H es el núcleo.
- La *Head Feature Convention* es un principio general que determina que la categoría de la proyección intermedia y de las proyecciones máximas deben coincidir con la del núcleo. De este modo, en una regla de la forma “ $\text{V}^2 \rightarrow \text{H}$ ”, H equivale a V
- Cuando un rasgo tiene la forma  $\langle \text{X}, +, - \rangle$  (i.e., es un rasgo binario), se lo puede simplificar como  $+\text{X}$  o  $-\text{X}$ .

El rasgo SUBCAT permite agrupar los distintos patrones de subcategorización.

Ver  
ejercicio  
en 5.3.4

- $\text{VP} \rightarrow \text{H}_{[1]}$  (ejemplo: *die*)
- $\text{VP} \rightarrow \text{H}_{[2]}, \text{NP}$  (ejemplo: *love*)
- $\text{VP} \rightarrow \text{H}_{[3]}, \text{NP}, \text{PP}_{[to]}$  (ejemplo: *give*)
- $\text{VP} \rightarrow \text{H}_{[4]}, \text{NP}, \text{PP}_{[for]}$  (ejemplo: *buy*)
- $\text{VP} \rightarrow \text{H}_{[5]}, \text{NP}, \text{NP}$  (ejemplo: *spare*)
- $\text{VP} \rightarrow \text{H}_{[6]}, \text{NP}, \text{PP}_{[+LOC]}$  (ejemplo: *put*)
- $\text{VP} \rightarrow \text{H}_{[8]}, \text{NP}, \text{S}[\text{FIN}]$  (ejemplo: *persuade*)

## 5 Gramática independiente de contexto generalizada (GPSG)

Las entradas léxicas son cuádruplas formadas por una forma fonológica, un conjunto de rasgos, información de irregularidad morfológica e información semántica<sup>4</sup>:

- (7)  $\langle \text{weep}, [[\text{CAT V}], [\text{BAR } 0], [\text{SUBCAT } 1]], \text{wept}, \text{weep}' \rangle$

A los fines de esta exposición, vamos a considerar solo los dos primeros elementos siguiendo la siguiente notación:

- (8)  $\langle \text{weep}, [[\text{CAT V}], [\text{BAR } 0], [\text{SUBCAT } 1]] \rangle$

### 5.2.3 Las metarreglas

Las metarreglas son funciones que mapean una regla ID en otra regla ID. Supongamos las oraciones de (9).

- (9) a. John bought the vegetables.  
b. Jim put the book on the table.  
c. Jim believed Bob to be the best.

Si nos concentramos en el VP, para generar estas estructuras se necesitan respectivamente las siguientes reglas ID

- (10) a.  $\text{VP} \rightarrow \text{H}_{[2]}, \text{NP}$  (Regla ID para el VP de 9a)  
b.  $\text{VP} \rightarrow \text{H}_{[6]}, \text{PP}_{[+LOC]}, \text{NP}$  (Regla ID para el VP de 9b)  
c.  $\text{VP} \rightarrow \text{H}_{[17]}, \text{VP}_{[INF]}, \text{NP}$  (Regla ID para el VP de 9c)

Supongamos ahora que queremos dar cuenta de cómo se forma el VP en las siguientes oraciones:

- (11) a. The vegetables were bought by John.  
b. The book was put on the table by Jim.  
c. Bob was believed by Jim to be the best.

Para eso necesitaríamos recurrir a las siguientes reglas:

---

<sup>4</sup>La comilla después de “weep” en su última aparición es para indicar que se trata de la interpretación semántica de *weep*.

- (12) a.  $VP[Pas] \rightarrow H_{[2]}, PP_{[BY]}$   
 b.  $VP[Pas] \rightarrow H_{[6]}, PP_{[+LOC]}, PP_{[BY]}$   
 c.  $VP[Pas] \rightarrow H_{[17]}, VP_{[INF]}, PP_{[BY]}$

Ahora, asumir estas reglas implica perder la relación de parentesco entre (11) y (9). Para dar cuenta de esto, se utiliza una metarregla como la siguiente:

- (13) **Metarregla pasiva:**  $VP \rightarrow \alpha, NP \Rightarrow VP[Pas] \rightarrow \alpha, (PP_{[BY]})$

Ahora, ya no es necesario estipular las reglas de (12), puesto que estas se siguen automáticamente de combinar las reglas de (10) con la metarregla de (13).

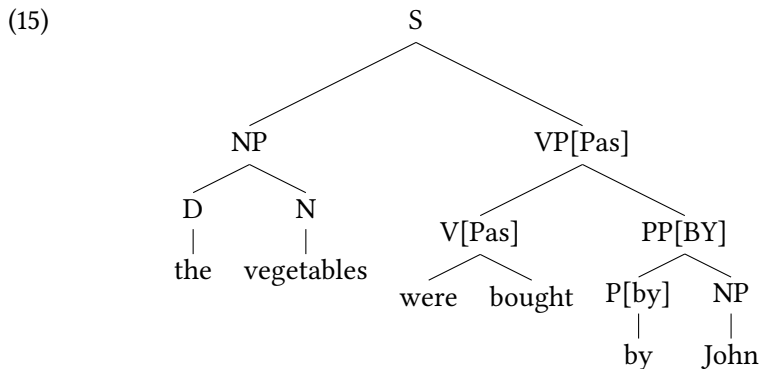
En Gazdar et al (1985) se reconocen 22 tipos de reglas de dominancia inmediata diferentes según el patrón de subcategorización del verbo. De estos 22, 10 son compatibles con la metarregla pasiva.

Ver  
ejercicio  
en 5.3.5

#### 5.2.4 Reglas léxicas

La metarregla nos permite reducir las reglas ID y establecer relaciones de parentesco entre reglas. Ahora bien, para introducir la forma activa o la forma pasiva del verbo, necesitamos entradas léxicas diferentes. Para resolver esto, GPSG recurre a reglas léxicas. Vamos a ver las reglas léxicas más en detalle más adelante en la cursada. Por el momento, supongamos que una regla léxica actúa del siguiente modo:

- (14) Si  $\alpha$  es un verbo activo,  $P(\alpha)$  es una perífrasis pasiva formada por el verbo ser conjugado más el participio pasivo de  $\alpha$ .



### 5.2.5 La semántica

GPSG asume una semántica en la línea de los trabajos de Montague. A diferencia de lo que vamos a ver cuando veamos implementaciones semánticas en gramáticas de rasgos, GPSG utiliza un sistema combinatorio orientado a teoría de tipos. La semántica orientada a tipos es la más frecuente en los trabajos de lingüística teórica, y la vamos a ver aplicada cuando lleguemos a la unidad de gramáticas categoriales. Por el momento, basta con ver que este tipo de abordajes semánticos utilizan recurrentemente modelos de las cosas que existen. El significado de las expresiones lingüísticas es su referencia, es decir, su correspondencia con elementos en estos modelos. Una teoría semántica involucra básicamente dos cuestiones:

1. especificar las denotaciones posibles para cada expresión de una categoría sintácticamente determinada, y
2. especificar la manera en la que las denotaciones de las expresiones complejas se producen en función de las denotaciones de sus constituyentes.

(Gazdar et al. 1985: 184)

El modelo está conformado por distintos tipos de cosas que se organizan en conjuntos. El modelo que asume GPSG consta de tres grandes tipos de elementos atómicos:

- Las entidades: Es el conjunto de todos los individuos, objetos y animales que existen en el mundo. Corresponde al tipo *e*.
- Los valores de verdad: Incluyen lo verdadero y lo falso. Corresponde al tipo *t*.
- Los mundos posibles: Es el conjunto de todos los modos en que las cosas pudieron ser. Permiten relativizar la verdad de las oraciones a determinados mundos (e.g., *Juan Peterson es un médium* es falsa en el mundo actual  $w_1$ , pero verdadero en el mundo  $w_2$  en que transcurre *Nuestra parte de noche* de Mariana Enríquez. Corresponde al tipo *s*.

En otros términos:

- (16)
- a.  $D_e = \{x: x \text{ es una entidad existente en el modelo}\}$
  - b.  $D_t = \{0, 1\}$
  - c.  $D_s = \{w: w \text{ es un mundo posible}\}$

En su exposición, Müller 2016 obvia los mundos posibles. Vamos a hacer lo mismo.

A partir de los tipos simples, es posible formar recursivamente tipos complejos. Por ejemplo, un verbo transitivo es de tipo  $\langle e, t \rangle$ , porque es una función que toma una entidad y devuelve un valor verdad. GPSG, en una versión simplificada, asume los siguientes tipos para algunas de las expresiones lingüísticas:

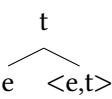
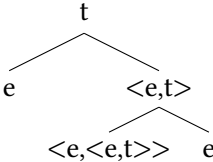
- (17) a.  $TYP(S) = t$   
 b.  $TYP(NP) = \langle \langle e, t \rangle, t \rangle$   
 c.  $TYP(N') = \langle e, t \rangle$   
 d.  $TYP(Det) = \langle TYP(N'), TYP(NP) \rangle$   
 e.  $TYP(VP) = \langle e, t \rangle$

Existe una regla general para obtener una denotación a partir de las denotaciones de dos expresiones lingüísticas que se combinan: Aplicación funcional

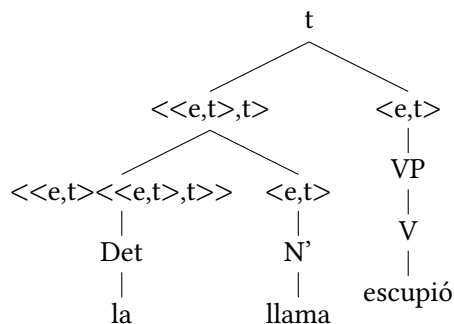
- (18) **Aplicación funcional:** Si  $\alpha$  es de tipo  $\langle b, a \rangle$  y  $\beta$  es de tipo  $b$ , entonces  $\alpha(\beta)$  es de tipo  $a$ .

Para poder aplicar esta regla de manera generalizada, GPSG estipula ciertas reglas que hacen que las reglas ID que no sean binarias, como la que reescribe a los verbos ditransitivos, acaben siendo binarias en la práctica. De este modo, al utilizar aplicación funcional, se obtienen árboles como los que se especifican a continuación.

Ver  
ejercicio  
en 5.3.6

- (19) a. 
- b. 

c.



De este modo, se usa teoría de conjuntos y funciones para modelar el significado de las expresiones lingüísticas y una regla básica que permite deducir el significado de una expresión compleja a partir de sus partes constituyentes, la regla de aplicación funcional en (18).

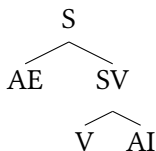
### 5.2.6 Los postulados de significado

El recurso de la metarregla permite relacionar dos formas. Pero no relaciona en sí mismo el significado para estas formas. Para eso vamos a agregar al sistema formal un mecanismo que se conoce como postulado de significado, que permite establecer sinonimia entre dos formas. Una versión absolutamente simplificada e informal del postulado de significado pasivo sería:

- The first nominal sister of a passive verb is interpreted as the last nominal argument of the active verb; the last nominal sister of a passive verb is interpreted as the first argument of the active verb. (Bennet 1995: 140)

En términos más formales, esto significa que si el verbo activo  $V$  es una función que toma dos entidades  $x$  e  $y$  en ese orden (siendo  $x$  el argumento interno e  $y$  el argumento externo, por lo tanto, más lejano, como se ve en (20)) y devuelve como resultado  $V(x,y)$ , en donde el primer elemento es el agente y el segundo el tema, la traducción semántica de  $V$  en  $V[\text{Pas}]$  mediante este postulado de significado lo convierte en una función que ahora toma  $y$  y  $x$  en ese orden y devuelve  $V[\text{Pas}](x,y)$ .

(20)



### 5.2.7 Las dependencias no acotadas

Supongamos que queremos dar cuenta de la siguiente oración, que, como vimos, parece obedecer a una estructura que no puede ser generada por una gramática independiente de contexto.

(21) Qué dijiste que vio.

GPSG va a hacer uso de las herramientas que vimos hasta ahora más un principio adicional, el Principio del rasgo Foot (FFP por sus siglas en inglés), para dar cuenta de esta oración sin violentar el formato de reglas esperado para una gramática independiente de contexto. Supongamos que tenemos la siguiente gramática

(22) **Reglas para un fragmento de GPSG2**

- a.  $VP \rightarrow V[2], NP[-Null]$
- b.  $VP \rightarrow V[8], S[+FIN, <COMP \text{ que}>]$
- c.  $S[+FIN, <COMP \text{ que}>] \rightarrow \text{que}, S$
- d.  $S \rightarrow NP[-Q] VP$

(23) **Entradas léxicas para un fragmento de GPSG2**

- a.  $<\text{dijiste}, V[8]>$
- b.  $<\text{vio}, V[2]>$
- c.  $<\text{caballos}, NP[-Q, -Null]>$
- d.  $<\epsilon, N[-Q, +Null]>$
- e.  $<\text{qué}, NP[+Q, -Null]>$
- f.  $<\text{que}, C>$

Suponiendo ciertas reglas LP, esta gramática genera la siguiente oración:

(24) Dijiste que vio caballos

Para generar dependencias cruzadas como la de (21), podemos utilizar una metarregla que convierte a toda regla de reescritura del siguiente modo<sup>5</sup>:

(25) **Metarregla para introducir preguntas parciales:**

$VP \rightarrow W, X \Rightarrow VP \rightarrow W, X[+Null]/X[+Q]$

---

<sup>5</sup>En el modelo más estándar presentado en Gazdar et al. 1985, se utiliza un principio general en lugar de una metarregla. Vamos a seguir en este punto a Müller 2016, que retoma una metarregla que se había postulado en los inicios de la teoría

W es, como en el caso de la metarregla pasiva, una variable. Por la HFC, debe incluir un H de categoría V. Esta metarregla nos permite traducir cualquier regla de nuestra gramática en una regla válida con slash. Aplicado a (22a) da como resultado la siguiente equivalencia:

$$(26) \quad VP \rightarrow V[2], NP[-Q] \Rightarrow VP \rightarrow V[2], NP[+Null]/NP[+Q]$$

Un principio denominado Principio del rasgo Foot determina que el nodo madre hereda los nodos del subconjunto que GPSG denomina rasgos Foot. Estos incluyen, por ejemplo, el rasgo Slash. Por este principio, entonces la regla que se da como resultado de (26) se traduce en los siguientes términos:

$$(27) \quad VP/NP[+Q] \rightarrow V[2], NP[+Null]/NP[+Q]$$

Esto, combinado con las entradas léxicas de (23) da lugar a la siguiente estructura:

$$(28) \quad \begin{array}{c} VP/NP[+Q] \\ \swarrow \quad \searrow \\ V[2] \quad NP[+Null]/NP[+Q] \\ | \quad \quad | \\ \text{vio} \quad \epsilon \end{array}$$

El Principio del rasgo Foot aplica hacia arriba en el árbol y da lugar a la siguiente estructura:

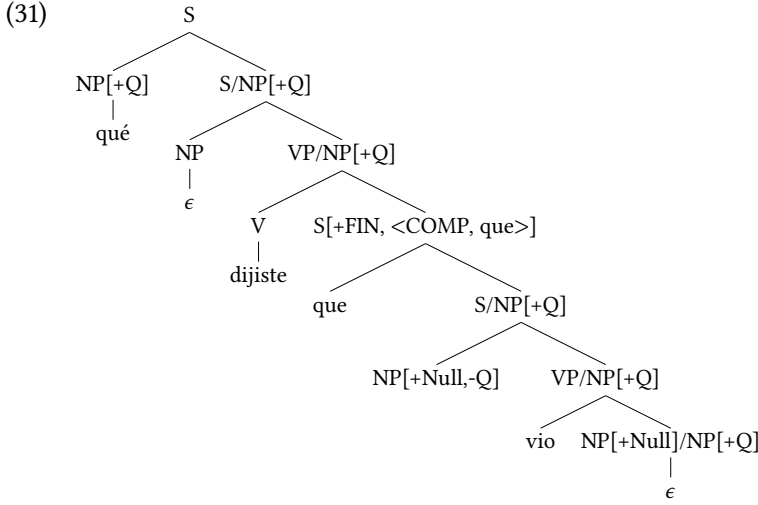
$$(29) \quad \begin{array}{c} S/NP[+Q] \\ \swarrow \quad \searrow \\ NP \quad VP/NP[+Q] \\ | \quad \quad \swarrow \quad \searrow \\ \epsilon \quad V \quad S[+FIN, <COMP, \text{que}>]/NP[+Q] \\ \quad | \quad \quad \swarrow \quad \searrow \\ \quad \text{dijiste} \quad \text{que} \quad S/NP[+Q] \\ \quad \quad \quad \swarrow \quad \searrow \\ \quad \quad NP[+Null, -Q] \quad VP/NP[+Q] \\ \quad \quad \quad \swarrow \quad \searrow \\ \quad \quad \text{vio} \quad NP[+Null]/NP[+Q] \\ \quad \quad \quad \quad | \\ \quad \quad \quad \quad \epsilon \end{array}$$



Supongamos ahora que tenemos una regla del siguiente tipo:

$$(30) \quad S \rightarrow X[+Q] \ S/X[+Q]$$

Al combinar esta regla con la derivación previa, se obtiene el siguiente árbol:



El rasgo Slash fue muy influyente en el campo de las gramáticas basadas en rasgos. De hecho, en unas clases vamos a ver una implementación de esto para una gramática enriquecida con rasgos en NLTK.

### 5.2.8 Sobre el poder expresivo de GPSG

GPSG, al conservar el formato de reglas de las gramáticas independientes y al contar con un número reducido de rasgos atómicos, es débilmente equivalente a una gramática independiente de contexto. Puesto que las gramáticas independientes de contexto no tienen el poder expresivo suficiente para dar cuenta del lenguaje natural, como se probó con posterioridad, eso hace que GPSG no sea una teoría descriptivamente adecuada.

Vimos en las clases previas que se sabe que es posible parsear una gramática independiente de contexto en tiempo polinómico. Como GPSG es débilmente equivalente a las Gramáticas Independientes de Contexto, se asumió que conservaban también un bajo costo computacional. Sin embargo, se ha demostrado que esto no es cierto: GPSG es probablemente intratable (ver Ristad 1986, 1987, 1990).

## 5 Gramática independiente de contexto generalizada (GPSG)

- En **Ristad 1986** se muestra que la fuente de este excesivo costo computacional proviene del uso de categorías vacías y de la convención del rasgo Head. El hecho de que GPSG lleve tanto tiempo de procesamiento a pesar de ser equivalente a una gramática independiente de contexto tiene que ver con el tamaño de la gramática. Normalmente, el costo de procesamiento se mide solo en términos de la longitud de la cadena, pero en realidad, el tamaño de una gramática también lo afecta. GPSG es equivalente a una gramática independiente de contexto lo suficientemente grande para que el impacto en el costo computacional sea relevante.
- Por otro lado, **Ristad 1990** muestra que el problema de la clausura finita de metarreglas (*i.e.*, el problema de determinar si una regla  $r$  pertenece al conjunto de las reglas formadas por la interacción de las metarreglas y las reglas ID de base) es demasiado complejo.
- **Ristad 1990** también demuestra que en GPSG el problema de reconocimiento universal para árboles locales no ordenados (*i.e.*, la posibilidad de decidir si una cadena  $x$  dada puede ser derivada de un conjunto de árboles locales no ordenados) es demasiado complejo.

Estas características hicieron que GPSG terminara perdiendo su atractivo. Por otro lado, GPSG se implementó en el lenguaje de programación PC-PATR (<https://software.sil.org/pc-patr/>). Este lenguaje de programación carece de soporte desde hace ya muchos años.

### 5.3 Ejercicios

#### 5.3.1 Reglas para gramática independientes de contexto

viene de ¿Cuáles de estas reglas pueden pertenecer a una Gramática Independiente de contexto y cuáles no?  
página 74

#### 5.3.2 Gramáticas independientes de contexto y linealidad

Viene de Supongamos las siguientes reglas de reescritura independientes de contexto:  
página 75

1.  $S \rightarrow SN SV$
2.  $SN \rightarrow N$
3.  $SV \rightarrow V SN_{[a]}$

4.  $SN_{[a]} \rightarrow a SN$

5.  $V \rightarrow AUX VLEX$

¿Cuáles de los siguientes órdenes son posibles?

1. AUX VLEX N a N

2. N AUX VLEX N a

3. N AUX VLEX a N

4. VLEX AUX N a N

Demuéstrelo proveyendo la derivación correspondiente.

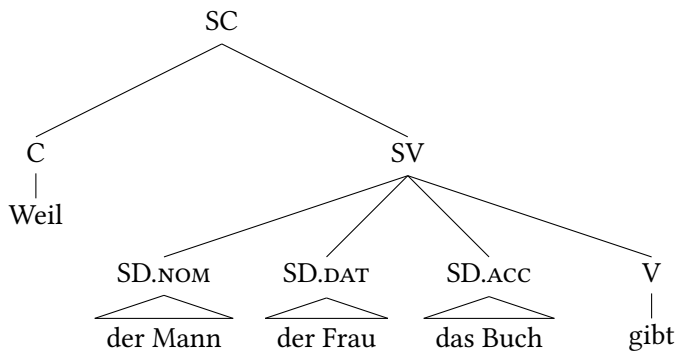
### 5.3.3 *Local reordering* en alemán

En alemán, los argumentos en el *Mittelfeld* pueden aparecer en cualquier orden, tal como se ve en el siguiente ejemplo. Este fenómeno se llama *local reordering*.

viene de  
página 75

Suponiendo el siguiente árbol para la primera oración, escriba una gramática independiente de contexto que dé cuenta de todos los órdenes. A los fines del ejercicio tome las subcadenas resumidas con triángulos como si se tratasen de un nodo terminal indivisible.

(32)



¿Qué problemas encuentra en esta formalización del *local reordering*?

#### 5.3.4 Un fragmento para GPSG (rasgo SUBCAT)

Viene de  
página 79

Elabore un fragmento de reglas de reescritura usando las convenciones de GPSG, particularmente la convención asociada al rasgo SUBCAT, que dé cuenta de los patrones de subcategorización en español que se ejemplifican a continuación:

(33) a. Marta se arrepentió de sus palabras.

- b. Marta murió.
- c. Maca le dio a Pablo el regalo.
- d. Maca compró un regalo.

Elabore el léxico como un par ordenado para los verbos arrepentir, morir, dar y comprar.

Agregue a su vez la entrada léxica para un verbo adicional por cada patrón de subcategorización ejemplificado arriba:

### 5.3.5 Metarregla y reglas ID

Viene de  
página 81

Observen la metarregla pasiva en (13), que repetimos acá a continuación:

(34) **Metarregla pasiva:**  $VP \rightarrow \alpha, NP_{[a]} \Rightarrow VP[Pas] \rightarrow \alpha, (PP_{[BY]})$

Cuáles de las siguientes reglas ID son compatibles con esta metarregla.

- (35)
- a.  $VP \rightarrow H[1]$
  - b.  $VP \rightarrow H[2], NP$
  - c.  $VP \rightarrow H[3], NP, PP[to]$
  - d.  $VP \rightarrow H[4], NP, PP[for]$
  - e.  $VP \rightarrow H[5], NP, NP$
  - f.  $VP \rightarrow H[6], NP, PP[ + LOC]$
  - g.  $VP[ + AUX] \rightarrow H[7], XP[+PRD]$
  - h.  $VP \rightarrow H[8], NP, S[FIN]$
  - i.  $VP \rightarrow H[9], (PP[to]), S[FIN]$
  - j.  $VP \rightarrow H[10], S[BSE]$
  - k.  $VP \rightarrow H[11], (PP[of]), S[BSE]$
  - l.  $VP[INF, + AUX] \rightarrow H[12], VP[BSE]$
  - m.  $VP \rightarrow H[13], VP[INF]$
  - n.  $VP \rightarrow H[14], V2[INF, + NORM]$
  - o.  $VP \rightarrow H[15], VP[INF, + NORM]$

Ahora, tome en cuenta la metarregla de adjuntos:

(36)  $VP \rightarrow W \Rightarrow VP \rightarrow W, AdvP^*$

Cuáles de las siguientes reglas son compatibles con esa metarregla:

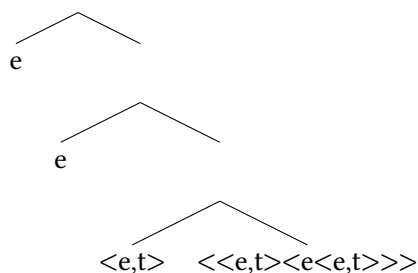
- (37)
- a.  $VP \rightarrow H[1]$
  - b.  $A^1 \rightarrow H[24], PP[about]$
  - c.  $VP \rightarrow H[2], NP$
  - d.  $N^1 \rightarrow H[30]$
  - e.  $VP \rightarrow H[3], NP, PP[to]$
  - f.  $A^1[AGR S] \rightarrow H[25], PP[to]$
  - g.  $VP \rightarrow H[4], NP, PP[for]$
  - h.  $N^1 \rightarrow H[31], PP[with], PV[about]$
  - i.  $N^1 \rightarrow H[32], S[COMP that]$
  - j.  $VP \rightarrow H[5], NP, NP$
  - k.  $A^1 \rightarrow H[26], S[FIN]$
  - l.  $VP \rightarrow H[6], NP, PP[+LOC]$

### 5.3.6 Aplicación funcional

Determine el tipo que se obtiene en los nodos faltantes al utilizar en todos los casos aplicación funcional:

Viene de la  
página 83

(38) a.







## **Part III**

# **Gramáticas basadas en constituyentes**



## **6 Gramáticas minimalistas**



# References

- Bach, Emmon W. 1989. *Informal Lectures on Formal Semantics*. Albany: State University of New York Press.
- Bennet, Paul. 1995. *A Course in Generalized Phrase Structure Grammar*. London: University College London.
- Bloomfield, Leonard. 1933. *Language*. New York: Holt, Rinehart & Winston.
- Bresnan, Joan (ed.). 1982. *The Mental Representation of Grammatical Relations*. Cambridge: The MIT Press.
- Chomsky, Noam. 1957. *Syntactic structures*. Manejamos la traducción de Carlos Peregrín Otero, Madrid: Siglo XXI, 1982. The Hague: Mouton.
- Chomsky, Noam. 1963. Formal properties of grammars. In Robert Duncan Luce, Robert R. Bush & Eugene Galanter (eds.), *Handbook of Mathematical Psychology: Volume II*, 323–418. New York, London: John Wiley & sons.
- Chomsky, Noam. 1965. *Aspects of the theory of syntax*. Manejamos la traducción de Carlos Peregrín Otero, Madrid: Aguilar, 1971. Cambridge, Massachusetts: M.I.T. Press.
- Chomsky, Noam. 1990. On formalization and formal linguistics. *Natural language & Linguistic theory* 8. 143–147.
- Chomsky, Noam & George Miller. 1963. Formal analysis of natural languages. In Robert Duncan Luce, Robert R. Bush & Eugene Galanter (eds.), *Handbook of mathematical psychology: volume ii*, 269–321. New York, London: John Wiley & sons.
- Di Tullio, Ángela. 1990. Lineamientos para una nueva gramática pedagógica. *Revista de Lengua y Literatura* 4(8). 3–12.
- Earley, Jay. 1970. An efficient context-free parsing algorithm. *Communication of the ACM* 13(2).
- Gazdar, Gerald. 1982. Phrase Structure Grammar. In Pauline Jacobson & Geoffrey Pullum (eds.), *The Nature of Syntactic Representation*, 131–186. Dordrecht: Reidel.
- Gazdar, Gerald, Ewan Klein, Geoffrey Pullum & Ivan Sag. 1985. *Generalized phrase structure grammar*. Cambridge, Massachusetts: Harvard University Press.
- Graham, Susan L & Michael A Harrison. 1976. Parsing of general Context-Free Languages. *Advances in Computers* 14. 77–185.

## References

- Hopcroft, John E, Rajeev Motwani & Jeffrey D Ullman. 2006. *Automata theory, languages, and computation*. Boston, Massachusetts: Addison-Wesley.
- Langacker, Ronald W. 1987. *Foundations of cognitive grammar: theoretical prerequisites*. Vol. 1. Stanford: Stanford university press.
- Langacker, Ronald W. 1991. *Foundations of Cognitive Grammar: Descriptive Application*. Vol. 2. Stanford: Stanford University Press.
- Martinet, André. 1984. *Elementos de lingüística general*. Edición original en francés, 1960. Madrid: Gredos.
- Moreno Sandoval, Antonio. 2001. *Gramáticas de unificación y rasgos*. Madrid: Antonio Machado.
- Müller, Stefan. 2016. *Grammatical theory: from transformational grammar to constraint-based approaches*. Berlin: Language Science Press.
- Partee, Barbara. 2016. Formal semantics. In Maria Aloni & Paul Dekker (eds.), *The Cambridge handbook of formal semantics*. Cambridge: Cambridge University Press.
- Partee, Barbara, Alice Meulen & Robert Wall. 2012. *Mathematical methods in linguistics*. Dordrecht: Kluwer Academics.
- Peregrín Otero, Carlos. 1970. *Introducción a la lingüística transformacional*. México DF: Siglo XXI.
- Peters, P. Stanley & Robert W. Ritchie. 1973. On the generative power of transformational grammars. *Information Sciences* 6. 49–83.
- Pollard, Carl & Ivan A Sag. 1994. *Head-driven phrase structure grammar*. Chicago, London: University of Chicago Press.
- Quesada, J. Daniel. 1974. *La lingüística generativo transformacional: supuestos e implicaciones*. Madrid: Alianza.
- Ristad, Eric Sven. 1986. Computational complexity of current gpsg theory. In *Proceedings of the 24th annual meeting on association for computational linguistics*, 30–39.
- Ristad, Eric Sven. 1987. Gpsg-recognition is np-hard. *Linguistic inquiry* 18(3). 530–536.
- Ristad, Eric Sven. 1990. Computational structure of gpsg models. *Linguistics and Philosophy* 13(5). 521–587.
- Roark, Brian & Richard Sproat. 2007. *Computational approaches to morphology and syntax*. Oxford: Oxford University Press.
- Saab, Andrés & Fernando Carranza. 2021. *Dimensiones del significado: una introducción a la semántica formal*. Buenos Aires: SADAF.
- Shieber, Stuart. 1986. *An introduction to unification-based theories of grammar*. Stanford: CSLI Lecture Notes.
- Solias Arís, María Teresa. 2015. *Métodos formales en lingüística*. Síntesis.

- Uszkoreit, Hans. 1987. *Word order and constituent structure in german*. Center for the Study of Language & Information.
- Valiant, Leslie G. 1975. General context-free recognition in less than cubic time. *Journal of computer and system sciences* 10. 308–315.
- Wintner, Shuly. 2010. Formal language theory. In Alexander Clark, Chris Fox & Shalom Lappin (eds.), *The handbook of computational linguistics and natural language processing*, 11–42. Massachusetts: Willey Blackwell.
- Yang, Yuan & Steven T Piantadosi. 2022. One model for the learning of language. *Proceedings of the National Academy of Sciences* 119(5).







