1. Recapitulación

En la clase pasada se trabajaron algunos conceptos clave que vamos a retomar hoy y que es importante tener presentes:

- **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).

- **Cadena**: secuencia de símbolos. En Python y Prolog corresponden al tipo "string" (en versiones previas de prolog se subsumía en el tipo "atom")

- **Lenguaje**: Conjunto de cadenas.

- **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.

- **Operaciones básicas de conjuntos**: Unión, intersección, diferencia, complemento.

- **Operaciones básicas sobre cadenas**: Concatenación, reversa.

Ver ejercicio en 5.1.

Todos estos conceptos forman parte de lo que se conoce como 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).

Bibliografía en la que está basada la clase de hoy:

- Bibliografía obligatoria
  - Quesada, J. D. (1974). *La lingüística generativo transformacional: supuestos e implicaciones*. Alianza, Madrid. "Capítulo 3: Las gramáticas generativas como sistemas formales", pp. 29–36.
  - Wintner, S. (2010). Formal language theory. En Clark, A., Fox, C., y Lappin, S., editores, *The Handbook of Computational Linguistics and Natural Language Processing*, pp. 11–42. Willey Blakwell, Massachusetts
  - Solias Arís, M. T. (2015). *Métodos formales en Lingüística*. Síntesis. Capítulo 2. "Las gramáticas formales", pp. 43–94.
- Roark, B. y Sproat, R. (2007). *Computational Approaches to Morphology and Syntax*. Oxford University Press, Oxford
- Hopcroft, J. E., Motwani, R., y Ullman, J. D. (2006). *Automata theory, languages, and computation*. Addison-Wesley, Boston, Massachusetts
- Partee, B., Meulen, A., y Wall, R. (2012). *Mathematical methods in linguistics*. Kluwer Academics, Dordrecht

2. Jerarquía de Chomsky

Dado un vocabulario $\{0, 1\}$, consideren los siguientes lenguajes posibles.

- **L1**: cualquier número de unos y ceros en cualquier orden
- **L2**: un número de unos equivalente al cuadrado del número de ceros que haya
- **L3**: un número cualquiera de unos seguidos de un número cualquiera de ceros
- **L4**: 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 *input*.
- **L5**: un número cualquiera de unos seguidos del mismo número de ceros.

Estos lenguajes difieren con respecto a la complejidad de la formulación de la descripción que los permite obtener. ¿Cómo los ordenarían si tuvieran que clasificarlos del más fácil de describir al más difícil?

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). La complejidad es directamente proporcional al 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 (se los denomina con el símbolo $\mathcal{L}$) que conforman la siguiente jerarquía (desde ya, estas clases están sujetas a subdivisiones internas).

- Lenguajes regulares
- Lenguajes independientes de contexto

- Lenguajes sensibles al contexto
- Lenguajes irrestrictos

Esto es lo que se conoce como **Jerarquía de Chomsky** La jerarquía de Chomsky está definida en términos de inclusión: Lenguajes regulares $\subset$ Lenguajes independientes de contexto $\subset$ Lenguajes sensibles al contexto $\subset$ Lenguajes irrestrictos.

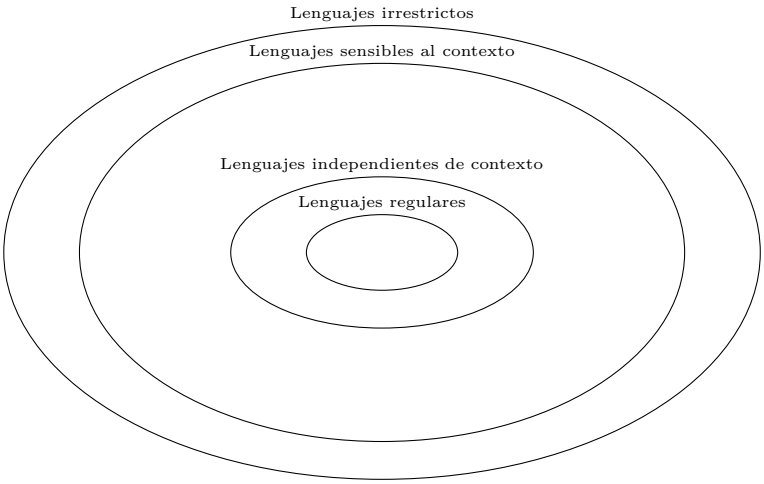


Figura 1: Diagrama de Venn de la jerarquía de lenguajes formales

Ver ejercicios en 5.2.

Así como es posible hacer operaciones entre conjuntos, también es posible hacer operaciones entre lenguajes, como unión, intersección y complemento. Asimismo, también es posible aplicar operaciones entre sus cadenas. Algunas de las más usuales son las siguientes:

- **Reverso**: Si w es una cadena, el reverso de w (w^R) también es una cadena.
- **Reverso**: 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.
- **Concatenación entre cadenas**: Si w_1 es una cadena y w_2 es otra cadena, la conacatenación entre w_1 y w_2 ($w_1 \cdot w_2$ o $w_1^\frown w_2$) también es una cadena

- **Concatenación entre lenguajes**: Si L1 y L2 son lenguajes, la concatenación entre ellos (L1·L2) es un lenguaje formado por la concatenación de cada cadena del primero con cada cadena del segundo.
- **Exponenciación**: Si L es un lenguaje $\{\alpha, \beta\}$, $L^0 = \{\epsilon\}$, $L^1 = \{\alpha, \beta\}$, $L^2 = \{\alpha\alpha, \alpha\beta, \beta\beta, \beta\alpha\}$;
- **Clausura de Kleene**: 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.

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, si L es un lenguaje regular, L^R también es un lenguaje regular.

Un problema sumamente relevante que involucra la relación entre lenguajes y cadenas es el problema de pertenencia

- **Problema de pertenencia o *fixed language recognition problem***: Dada una cadena w y un lenguaje L, ¿w ∈ L?
- **Problema de reconocimiento universal**: Dada una gramática G y una cadena L, ¿w ∈ L(G)

El problema del reconocimiento universal es el que tiene mayor peso en la teoría lingüística (Ristad 1986) y se aborda computacionalmente con dos clases de algoritmos:

- **Algoritmo de reconocimiento**: Algoritmo que chequea si determinada oración se sigue como teorema de determinada gramática (*i.e.*, si determinada cadena pertenece al lenguaje).
- **Algoritmo de *parsing***: A la vez de chequear si determinada oración se sigue como teorema de determinada gramática, devuelve en caso positivo una estructura para esa oración.

3. Formas de definir lenguajes

Convencionalmente, existen dos grandes formas de definir lenguajes por intensión: los autómatas y las gramáticas (formales).

3.1. Autómatas

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

- σ = Alfabeto
- Q = Conjunto de Estados
- q_0 = un único estado inicial
- F = un conjunto de estados finales
- δ = 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, pero se los suele representar mediante grafos. Supongamos un L6 definido del siguiente modo:

(1) a. L6 = {w: w es una cadena obtenida por la concatenación de a con n número de ocurrencias de b para cualquier n $\geq$0}

b. L6 = {a, ab, abb, abbb, abbbb...}

Se puede ver una posible representación mediante un grafo para el autómata A6 que describe el lenguaje L6 en la figura 2

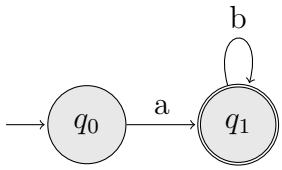


Figura 2: Autómata para A6

También se los puede representar mediante tablas de transición como la de la tabla 1.

Ver ejercicio 5.3

En (2) se define un autómata para este lenguaje L6 expresado como una quíntupla.

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

Tabla 1: Tabla de transición para A6

(2) a. A6 = ({a, b}, {q_0,q_1}, q_0, {q_1}, {<q_0, a, q_1 >, <q_0, b, $\emptyset$ >, <q_1, a, $\emptyset$ >, <q_1, b, q_1 >})

b. A6 = (Σ, Q, q_0, F, δ) donde Σ = {a, b}, Q = {q_0,q_1}, q_0, {q_1}, F = <q_1 y δ = {<q_0, a, q_1 >, <q_0, b, $\emptyset$ >, <q_1, a, $\emptyset$ >, <q_1, b, q_1 >}

El poder discriminativo de los autómatas depende fundamentalmente de la naturaleza de las instrucciones. 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 2.

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

Tabla 2: Equivalencia entre lenguajes y autómatas

3.2. Las gramáticas

Las gramáticas son sistemas deductivos formados por axiomas y reglas de deducción que permiten derivar las oraciones del lenguaje

natural como teoremas.

- **Axioma**: Una proposición básica que se da por supuesta de antemano y, por ende, no se demuestra.

- **Reglas de inferencia o reglas de deducción**: Un conjunto de reglas que se consideran válidas para derivar proposiciones a partir de otras.

- **Teorema**: Una proposición que se obtiene a partir de otras a partir de ciertas reglas válidas de inferencia o reglas de deducción. ´

Formalmente una gramática es una cuádrupla (V_T, V_N, S, R) del siguiente modo:

(3) 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.

(4) **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 = $\left\{\begin{array}{l} \text{R1. A} \rightarrow \text{aB} \\ \text{R2. B} \rightarrow \text{bB} \\ \text{R3. B} \rightarrow \text{b} \end{array}\right\}$

(5) **Derivación para abbb**

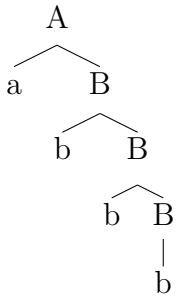
a. aB por R1

b. abB por R2

c. abbB por R2

d. abbb por R3

(6) **Estructura para abbb**



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

4. Clases de lenguajes

4.1. Clases de lenguajes por su capacidad discriminatoria

4.1.1. Lenguajes regulares

Expresiones regulares

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

Tabla 3: Equivalencia entre lenguajes, gramáticas y autómatas

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

(7) Dado un alfabeto Σ, el conjunto de expresiones regulares sobre Σ se define de la siguiente forma:

   a. $\emptyset$ es una expresión regular

   b. ϵ es una expresión regular

   c. Si a $\in \Sigma$, a es una expresión regular

   d. Si r1 y r2 son expresiones regulares, también lo son (r1 + r2) y (r1 $\cdot$ r2)

   e. Si r es una expresión regular, también lo es (r)*

   f. Nada más es una expresión regular

Los corchetes dobles $[\![\ ]\!]$ 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, $[\![r]\!]$ es el conjunto de cadenas que r describe.

(8) Dado un alfabeto Σ, el conjunto de expresiones regulares sobre Σ se define de la siguiente forma:

   a. $[\![\emptyset]\!] = \{\}$

   b. $[\![\epsilon]\!] = \{\epsilon\}$

c. Si a $\in \Sigma$, $[\![a]\!] = \{a\}$

d. Si r1 y r2 son expresiones regulares cuyas denotaciones son $[\![r1]\!]$ y $[\![r2]\!]$, $[\![(r1 + r2)]\!] = [\![r2]\!] \cup [\![r2]\!]$ y $[\![(r1 \cdot r2)]\!] = [\![r1]\!] \cdot [\![r2]\!]$

e. Si r es una expresión regular, $[\![(r)^*]\!] = [\![r]\!]^*$

La notación de arriba no es usual en la práctica. En la tabla 4 se especifican algunas de las convenciones notacionales que utiliza la librería re de Python.

Ver ejercicio en 5.4

Símbolo	**Significado**	**Ejemplo**
*	cero o más ocurrencias del caracter o bloque anterior	$[\![a^*]\!]$ = {w| w es una cadena formada por la concatenación de cero o más ocurrencias de a}
+	una o más ocurrencias del caracter o bloque anterior	$[\![a+]\!]$ = {w| w es una cadena formada por la concatenación de una o más ocurrencias de a}
?	una o ninguna ocurrencia del caracter o bloque anterior	$[\![ab?]\!]$ = {a, ab}
.	cualquier caracter excepto cambio de línea	$[\![a.]\!]$ = {aa, ab, ac, ad...}
{n}	n cantidad de ocurrencias del caracter o bloque anterior	$[\![a\{3\}]\!]$ = {aaa}

Símbolo	**Significado**	**Ejemplo**
{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)	⟦a{3,6}⟧ = {aaa, aaaa, aaaaa, aaaaaa}
[]	disyunción entre lo que aparezca adentro	⟦[abcd]⟧ = {a,b,c,d}
[x-y]	rango de caracteres que aparecen entre x e y	⟦[a-d]⟧ = {a,b,c,d}

Tabla 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 5 se resumen algunas de las más importantes.

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

Tabla 5: Abreviaturas para expresiones regulares siguiendo la librería re de Python (https://docs.python.org/3/library/re.html)

Los lenguajes regulares son cerrados ante las siguientes operaciones:

1. unión

2. concatenación

3. clausura de Kleene

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.

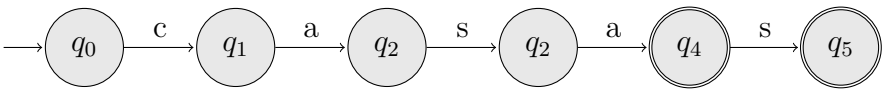


Figura 3: Autómata finito que reconoce el lenguaje {casa, casas}

Gramáticas regulares

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 (9), en donde X o Y deben leerse como símbolos no terminales (sean distintos o no) y x como un símbolo terminal:

ver ejercicio en 5.5

(9) a. X → x Y

b. X → x

Por ejemplo

(10) Gramática regular equivalente a la expresión regular a+

a. R1. A → a A

b. R2. A → a

(11) Derivación para aaaa

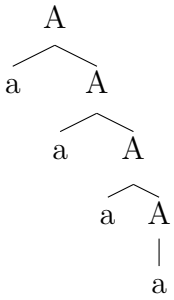
a. aA Por R1

b. aaA Por R1

c. aaaA Por R1

d. aaaa Por R2

(12) Árbol para la cadena aaaa



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 respectivas raíces.

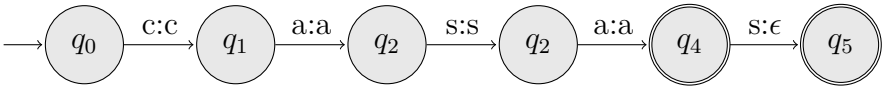


Figura 4: Transducer que devuelve el lema de casa o casas

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:

(13) {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.

(14) Gramática Independiente de Contexto

a. V_T = símbolos terminales = {a, b, c, d, e}

b. V_N = símbolos no terminales = {A, B}

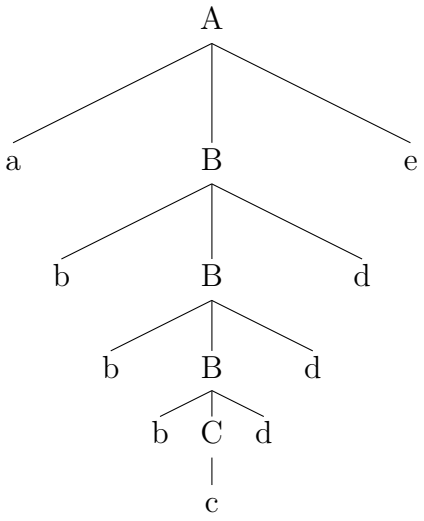
c. S = el axioma inicial, incluido en V_N = A

d. R = conjunto de reglas = $\begin{Bmatrix} \text{R1. A} \rightarrow \text{aBe} \\ \text{R2. B} \rightarrow \text{cBd} \\ \text{R3. B} \rightarrow \text{cCd} \\ \text{R4. C} \rightarrow \text{c} \end{Bmatrix}$

(15) Derivación para abbbcddde

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

(16)



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*).

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.

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 –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).

4.3. 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 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 5 se comparan las rsepectivas curvas.

Se sabe que el procedimiento para parsear una cadena según una gramática tiene un costo de procesamiento según la tabla 7.

La escala de complejidad nos permite saber qué clase de problemas podemos tratar de resolver en una computadora, cuáles solo

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!)

Tabla 6: Tipos de costos de procesamiento medidos en términos temporales y notación de O mayúscula

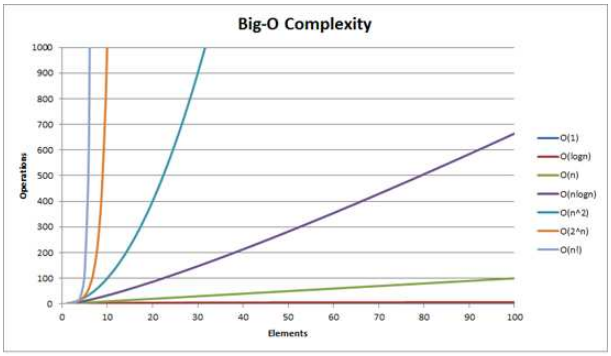


Figura 5: Comparación de curvas de los distintos costos de procesamiento (tomado de https://i.stack.imgur.com/Aq09a.png)

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	indecidible

Tabla 7: Costos de procesamiento en términos de tiempo para el parseo de una cadena según el tipo de gramática

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 inde-

pendientes 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, 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 | w es una cadena de ceros y unos y la traducción de w en una máquina de Turing M acepta a w como input, es decir w ∈ 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.

5. Ejercicios

5.1. Lenguajes y fenómenos lingüísticos

Viene de página 2

Identifique cuál de estos lenguajes caracteriza mejor las cadenas formadas por la concategnación de las subcadenas subrayadas y justifique[1].

[1]Para una aplicación práctica de este tipo de ejercicio, ver Yang y Piantadosi (2022). Entiéndase por a{n}b{n} una cadena de n ocurrencias de a seguida de n ocurrencias de b.

(17) 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.

(18) a. El mono

b. El mono adorable

c. El mono adorable y amistoso

d. El mono adorable, amistoso y joven.

(19) 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.

(20) 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 que sea providencial deje de sorprenderme deje de sorprenderme deje de sorprenderme

5.2. Operaciones sobre lenguajes

Viene de página 4

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

(21) a. L1 = {hola, chau, buenas}

b. L2 = {a,b}

1. L2∪L1 =

2. L1∩L2 =

3. $L1^R$ =

4. $L1^1$ =

5. L2·L1 =

6. $L2^3 =$

7. $L2^R =$

5.3. Autómatas

Observe el autómata de la figura 6 para el lenguaje L7

Viene de página 6

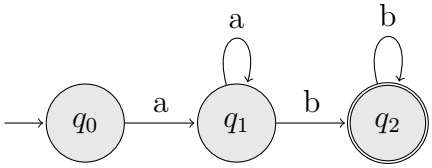


Figura 6: Autómata para L7

Defina por intensión el conjunto L7

(22) $L7 =$

Defina el autómata como una quíntupla:

(23) a. $A7 =$

b. $A7 =$

Complete el cuadro de transición de estados en la tabla 8.

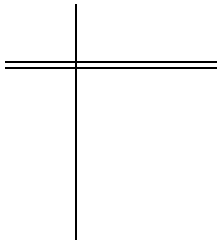


Tabla 8: Tabla de transición para A7

5.4. Expresiones regulares

5.4.1. Denotación de expresiones regulares

Viene de página 11

Escriba las denotaciones para las siguientes expresiones regulares:

(24) a. ⟦ab?c[da]⟧ =

b. ⟦[Dd]ra?\.⟧ =

c. ⟦\d{2}(\d{2})⟧ =

d. ⟦\w\s⟧ =

5.4.2. Armado de expresiones regulares

Escriba expresiones regulares para los siguientes tipos de cadenas

(25) a. Palabras (cadenas separadas por espacios) =

b. Links de internet =

c. Oraciones (cadenas que terminen en un punto) =

5.5. Gramáticas regulares

Suponga la siguiente gramática:

Viene de la página 13

(26) **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 = $\begin{Bmatrix} \text{R1. A} \rightarrow \text{aA} \\ \text{R2. A} \rightarrow \text{aB} \\ \text{R3. B} \rightarrow \text{b} \\ \text{R4. B} \rightarrow \text{c} \end{Bmatrix}$

(27) **Derivación para aaab**

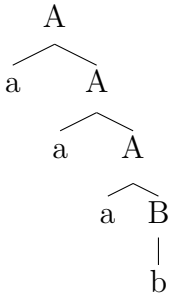
a. aA por R1

b. aaA por R1

c. aaaB por R2

d. aaab por R3

(28) **Estructura para aaab**



Escriba la expresión regular equivalente a este lenguaje:

5.6. Tipos de reglas gramaticales

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

(29) 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. $A\ b \rightarrow B$

5.7. Repaso

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

(30) 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 Σ^*

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 leguaje en cuestión.

f. Los autómatas generan fuertemente estructuras.

g. El complemento de un lenguaje recursivo también es necesariamente un lenguaje recursivo

Referencias

Chomsky, N. (1963). Formal properties of grammars. En Luce, R. D., Bush, R. R., y Galanter, E., editores, *Handbook of Mathematical Psychology: Volume II*, pp. 323–418. John Wiley and sons, New York, London.

Chomsky, N. y Miller, G. (1963). Formal analysis of natural languages. En Luce, R. D., Bush, R. R., y Galanter, E., editores, *Handbook of Mathematical Psychology: Volume II*, pp. 269–321. John Wiley and sons, New York, London.

Hopcroft, J. E., Motwani, R., y Ullman, J. D. (2006). *Automata theory, languages, and computation*. Addison-Wesley, Boston, Massachusetts.

Partee, B., Meulen, A., y Wall, R. (2012). *Mathematical methods in linguistics*. Kluwer Academics, Dordrecht.

Quesada, J. D. (1974). *La lingüística generativo transformacional: supuestos e implicaciones*. Alianza, Madrid.

Ristad, E. S. (1986). Computational complexity of current gpsg theory. En *Proceedings of the 24th annual meeting on Association for Computational Linguistics*, pp. 30–39. Association for Computational Linguistics.

Roark, B. y Sproat, R. (2007). *Computational Approaches to Morphology and Syntax*. Oxford University Press, Oxford.

Solias Arís, M. T. (2015). *Métodos formales en Lingüística*. Síntesis.

Wintner, S. (2010). Formal language theory. En Clark, A., Fox, C., y Lappin, S., editores, *The Handbook of Computational Linguistics and Natural Language Processing*, pp. 11–42. Willey Blakwell, Massachusetts.

Yang, Y. y Piantadosi, S. T. (2022). One model for the learning of language. *Proceedings of the National Academy of Sciences*, 119(5).