"Lenguajes Formales" (para Ingenieros Informáticos)

Domingo Gómez & Luis M. Pardo & Cristina Tîrnăucă

15 de noviembre de 2013

Prólogo

> ¡Mantened la calma! ¡No os dejeis llevar por el pánico!
>
> Primera página de la Guia del Autoestopista Intergaláctico

El origen de estos apuntes son las notas tomadas para la asignatura "Teoría de Autómatas y Lenguajes Formales" que se comenzó a impartir en el curso 2007/08. La evolución de los contenidos ha sido siempre en función de las diversas promociones, su formación previa y su capacidad de adquirir nuevos conocimientos. Estos apuntes representan una versión definitiva de nuestra experiencia. Nuestro cuidado ha sido conseguir una adquisición de conocimientos teóricos, su aplicación con una formación de carácter matemático para que el alumno desarrolle una actitud rigurosa ante problemas de ésta y otras materias.

Este aspecto es de gran importancia en una materia como los lenguajes formales, considerados como uno de los fundamentos de las ciencias de computación. Dada la escasa formación previa en matemáticas de los alumnos que llegan a la universidad española, se ha pretendido mantener el máximo de formalismo en definiciones y enunciados pero renunciando a las demostraciones en términos formales completos, y limitando éstas a aquellos casos en los que la prueba se basa en la existencia de un algoritmo. El objetivo del curso es que el alumno comprenda, aplique y asimile una serie de herramientas matemáticas que se han desarrollado para la teoría de lenguajes formales.

Estos apuntes están también pensados para "aprender a resolver problemas algorítimicamente". Según nuestro punto de vista, la mejor forma para conseguir este objetivo es adquirir experiencia en la resolución de problemas a través de algoritmos definidos con precisión.

Debido a lo mencionado, nuestro planteamiento será constructivista en el mejor sentido de la palabra. Estos apuntes están dirigidos a futuros ingenieros, por lo que intentaremos que los algoritmos presentados cumplan criterios de eficiencia y estén orientados a su implementación[1] en una aplicación informática.

Con esto esperamos que, aunque la teoría de la computación sea un área abstracta dentro de la ingeniería informática, el alumno comprenda la utilidad de esta rama, pese a su complejidad. Por ello incluimos ejemplos, cuestiones y problemas que varían desde preguntas triviales hasta retos que requerirán expresar lo aprendido con argumentos semi–formales.

Los autores han elaborado este libro para una asignatura cuatrimestral. Su objetivo ha sido que los alumnos conocieran, a la finalización de la asignatura los siguientes conceptos,

- lenguajes regulares y autómatas finitos,
- lenguajes libres de contexto y autómatas con pila.

[1] al menos en los casos en que ésto sea posible

A partir estos conocimientos, se espera que adquieran habilidades que incluyan generar autómatas que reconozcan lenguajes y los algoritmos que permiten pasar de autómatas a gramáticas y viceversa.
Las referencias bibliográficas no pretenden ser exhaustivas, conteniendo lo esencial de la literatura en estos temas y dejando abierta la posibilidad de investigar al lector interesado.

Índice general

Capítulo 1

Primeros pasos con los lenguajes formales

Índice

1.1. Introducción

Here be dragons

Hamlet

En estos apuntes intentaremos dar una pequeña introducción a cada tema con una frase que ponga en antecedentes al lector de estos apuntes. Esta frase de Hamlet se utiliza para expresar cuando se entra en terreno desconocido. Esta materia da los primeros pasos en lo que es conocido como la teoría de computación, esto es preguntarnos que es lo que se puede computar a partir de calculos automáticos, y una de las partes importantes en los primeros pasos será entender como representar los cálculos de forma autómatica. Para ello, empezaremos discutiendo sobre que es un modelo de cálculo.
Cuando se fija un modelo de cálculo, se hace referencia a los fundamentos de la comunicación y el lenguaje. Fijado el modelo de cálculo, podemos también preguntarnos por que algoritmos podemos utilizar en ese modelo de calculo. Informalmente, un algoritmo es cualquier procedimiento estructurado que tome unos datos de entrada y retorne unos valores, la salida. Generalizaremos todo cálculo algorítmico como un proceso de comunicación: algo es emitido (la entrada), manipulado y respondido (la salida). Es una comunicación entre hombre y máquina o una comunicación entre seres humanos. Pero el principio de esta discusión debe orientarse hacia lo

que es susceptible de ser comunicado (tanto la entrada como la salida son objetos comunicables).
Nos vamos directamente al Círculo de Viena, con precursores como K. Popper, y con actores activos como R. Carnap, H. Hahn y O. Neurath. En el ámbito de la lógica matemática son relevantes la pertenencia de miembros de la talla de K. Gödel o A. Tarski. Este influyente conjunto de filósofos, matemáticos y lógicos se ve roto por el nazismo en pedazos incomponibles, pero influyó muy notablemente la filosofía y la lógica de primeros de siglo (hasta finales de los 30).
Apliquemos el empirismo lógico como ideología provisional. Tomemos la disquisición inicial: ¿qué es susceptible de ser comunicado?.
Una respuesta razonable (empírica en nuestra aproximación) es que es comunicable todo aquello expresable en un alfabeto finito. Nuestra aproximación empírica se basa en la experiencia: no conozco ningún caso de información emitida o recibida por alguien, con significado claro y único, que no haya sido expresada sobre un alfabeto finito.
A partir de esta idea consideraremos como Σ un conjunto finito que denominaremos *alfabeto* y por Σ^*[1] el conjunto de todas las *palabras* expresables sobre este alfabeto finito.
Dos precisiones importantes: Comunicado (el significante) es una palabra sobre un alfabeto finito, pero el significado (la componente semántica de la comunicación) no es tan claramente finito. Tomemos un ejemplo de las matemáticas. Sea D^1 el conjunto de números reales dado por:

$$\{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 - 1 \leq 0\}$$

El tal conjunto no es finito, ni contable. Podría quizá discutirse su existencia (uno de los problemas más difíciles de la filosofía de las matemáticas es determinar el significado de existencia: existe lo que es expresable –esto es seguro–, pero, ¿existe lo que no puedo expresar?[2]). Suponiendo que $\mathbb{R}$ exista, yo puedo expresar un conjunto cuyo cardinal no es numerable mediante una expresión sobre un alfabeto finito. Por lo tanto, los significantes caminan sobre una digitalización finita, sobre un alfabeto finito, no así los significados. Lo que implica que mientras es fácil representar las palabras, es difícil representar los posibles significados. Olvidemos, finalmente, que la modelización continua de la semántica es una de las corrientes de última moda; pero tampoco olvidemos que la semántica (y la Semiótica) cuentan con los elementos adicionales de la subjetividad que son bastante "difusos".
La segunda consideración es que nosotros usaremos el lenguaje de la Teoría de la Recursividad y no el de la Lingüística. Para referencias al asunto véase, por ejemplo, [Marcus, 67]. En este caso, la terminología se modifica del modo siguiente: el alfabeto se denomina vocabulario, las palabras son lo mismo, y el lenguaje es una cantidad, posiblemente infinita, de palabras sobre el vocabulario. Pero vayamos a nuestra definición:

[1] Veremos más adelante que significa la notación estrella.

[2] Esto será lo máximo que nos introduciremos por los oscuros caminos de la filosofía. Paul Gordan, gran matemático del siglo XIX, amonestó a David Hilbert con su famosa frase "Das ist keine Matematik, Das ist Theologie" por "demostrar" la existencia de objetos, sin "mostrar" esos objetos.

Definición 1 (Alfabeto) *Sea Σ un conjunto finito que llamaremos alfabeto.*

- *Una palabra sobre Σ es una lista finita de símbolos de Σ. Podemos formalmente identificar las listas $x = x_1 \cdots x_n$ de símbolos ($x_i \in \Sigma$) con los elementos del producto cartesiano Σ^n. Denotaremos por $\mid x \mid = n$ la longitud de la palabra $x_1 \cdots x_n$.*
- *El conjunto de todas las palabras sobre el alfabeto Σ se denotará mediante Σ^* y podemos identificarlo con la unión disjunta*

$$\Sigma^* = \dot{\bigcup_{n\in\mathbb{N}}} \Sigma^n$$

- *Los subconjuntos L de Σ^* se denominan lenguajes.*

Insistamos en la notación $x_1 \cdots x_n$ para expresar la palabra $(x_1, \ldots, x_n) \in \Sigma^n$. Los ")" y las ", " pudieran ser (y suelen ser) elementos del alfabeto.
Tomemos la letra a y nótese la identificación obvia, cuando $\Sigma = \{a\}$ es un alfabeto de una sola palabra, entre Σ^* y $\mathbb{N}$.

Nota 2 *La única observación relevante es que si Σ es un conjunto finito Σ^* es un conjunto numerable. No podemos expresar mucho más allá que una cantidad numerable de significantes (a lo sumo). La verdad, no es gran cosa: una sonata de Mozart no es sino una triste palabra de unas pocas numerables posibles.*
El considerar alfabetos numerables no cambiaría gran cosa, lo que se puede expresar sobre un alfabeto numerable es expresable sobre un alfabeto finito (por razones obvias). El punto duro comienza cuando uno se pregunta si es posible la comunicación (hombre–máquina u hombre–hombre) a través de lenguajes sobre alfabetos no numerables.

Otros ejemplos, "El Quijote", entendido como el libro completo, es, para nuestro contexto, una palabra sobre el alfabeto del castellano, i.e. $\{a, b, \ldots, z\}$, $\{A, B, \ldots, Z\}$, {?,Á, !,",","."," ̇:,"♯",.}, donde las "," y "." son los obvios, ̇: es el "punto-y-aparte" y ♯ son los espacios entre palabras.
Uno podría muy bien argumentar porqué el "Otello" de Shakespeare no es una palabra del castellano y la respuesta es la obvia: es una palabra sobre el alfabeto castellano; pero el castellano no es solamente un alfabeto, sino un lenguaje $\mathcal{C} \subseteq \Sigma^*$ en el que se incluyen solamente las palabras formadas por sucesiones de símbolos del Diccionario de la Real Academia de la Lengua (ver autómatas finitos para más disquisiciones). El "Otello" pertenece al lenguaje inglés $\mathcal{I} \subseteq \Sigma^*$. Una versión original de la Ilíada[3], el Corán, El Idiota o los Vedas no son subconjuntos de Σ^* por usar diferentes alfabetos (el griego, el árabe, el cirílico o el sánscrito). Por esta razón y durante el transcurso del libro, variarán tanto los alfabetos como los lenguajes. Aunque la comunicación occidental tenga en común el alfabeto, no hace más sencillo el problema de comunicarse utilizando diferentes lenguajes. Sin embargo, las traducciones muestran que no hay mucho que añadir aunque se cambie el alfabeto.

[3] Aunque la tradición mantiene la ceguera de Homero y, por tanto, la transmisión oral de sus versos, aceptamos como "original" cualquier versión escrita durante el período helenístico.

Esto muestra que alfabetos complicados o sencillos no tienen relación con la simplicidad del lenguaje. Aprovecharemos este momento para introducir un lenguaje que volverá a aparecer más adelante.

Ejemplo 1 *Sea* $\Sigma := \{ A, C, G, T \}$*, representando las cuatro bases que conforman el ADN, a saber: Adenina, Guanina, Citosina y Timina. Las cadenas de ADN forman un lenguaje que contiene la información genética y esta empaquetada de esta forma para su posible transmisión hereditaria. Curiosamente, este lenguaje es casi universal y se aplica a todos los seres vivos menos en excepciones contadas dentro de casos contados.*

Cada cadena de ADN guarda la codificación de varias proteínas. Dentro del ADN, cada secuencia de tres bases de las mencionadas corresponden a un aminoacido concreto. Esto se conoce como el código genético. Por ejemplo, la combinación ATG *representa el inicio de una secuencia y el final puede ser representada por* TGA, TAG, TAA[4] .

1.2. Lenguajes formales y monoides

> Conceptos sin intuiciones son vacíos, intuiciones sin conceptos son ciegas
>
> Inmanuel Kant

En esta sección intentaremos introducir las operaciones más elementales que se pueden realizar entre lenguajes, las cuales manejaremos frequentemente en este curso. Todas ellas son elementales, pero no simples y por ello nos detendremos algo más de lo necesario. Recomendamos encarecidamente que el lector se detenga también un poco más de lo necesario para hacer suyos estos conceptos.
La operación esencial sobre Σ^* es la concatenación (también llamada adjunción) de palabras:

$$\begin{aligned} \cdot : \Sigma^* \times \Sigma^* &\longrightarrow \Sigma^* \\ (x, y) &\longmapsto x \cdot y, \end{aligned}$$

es decir, si $x = x_1 \cdots x_n$ e $y = y_1 \cdots y_m$, entonces

$$x \cdot y = x_1 \cdots x_n \cdot y_1 \cdots y_m.$$

Denotemos por $\lambda \in \Sigma^*$ la palabra vacía (para distinguirla del lenguaje vacío $\emptyset$, usando la notación estándar de Teoría de Conjuntos.

Lema 3 $(\Sigma^*, \cdot)$ *es un monoide*[5] *, donde* λ *es el elemento neutro. La longitud define*

[4] El ADN siempre se ha considerado el "lenguaje de la vida" y parece que se cumple la máxima de Galileo: "La Naturaleza es un libro escrito con el lenguaje de las matemáticas".

[5] Recordemos que un monoide es un conjunto X con una operación $* : X \times X \longrightarrow X$ que verifica la propiedad asociativa y de tal modo que X contiene un elementos neutro.

un morfismo de monoides[6] *entre* Σ^* *y el conjunto de los número naturales. El monoide* Σ^* *es abeliano*[7] *si y solamente si el cardinal de* Σ *es uno.*

Demostración. Ejercicio obvio. ∎

Las propiedades de los monoides dan una estructura con la cual estudiar los lenguajes recursivamente enumerables. Por ejemplo, transformar la lengua en la cual expresamos un problema no añade dificultad a este. En general, veremos la forma de relacionar problemas con las transformaciones. Otra propiedad de los lenguajes es que las palabras dentro del lenguaje pueden ser ordenadas. En la demostración del siguiente lema se da una posible forma de como ordenar las palabras.

Lema 4 *Si* Σ *es un alfabeto finito, el conjunto* Σ^* *es numerable, esto es, es biyectable con el conjunto* $\mathbb{N}$ *de los números naturales.*

Demostración. Para dar una prueba de este enunciado basta con fijar un buen orden en Σ^*. Un ejercicio razonable consiste en definir el buen orden basado en "lexicográfico + longitud" que define la biyección. Recuérdese que el orden lexicográfico es el orden usual del "diccionario", i.e. basado en establecer un orden en el alfabeto Σ (en ocasiones lo llamarán alfabético). Es decir, sea $\ll$ un orden total en Σ (que existe por ser Σ finito). Supongamos que los elementos de Σ quedan ordenados mediante:

$$\Sigma := \{a_1 \ll \cdots \ll a_r\}.$$

Definimos para $y = y_1 \cdots y_m,\ x = x_1 \cdots x_n \in \Sigma^*$ la relación de orden siguiente:

$$x \leq y \Longleftrightarrow \begin{cases} \text{o bien} & n = |x| < |y| = m, & \\ \text{o bien} & |x| = |y| = n = m, & \begin{cases} \exists k \leq n = m, \forall i \leq k-1, \\ x_i = y_i, \\ x_k \ll y_k \end{cases} \\ \text{o bien} & x = y. & \end{cases}$$

Esta relación de orden es un buen orden en Σ^* y permite una identificación (biyección) entre Σ^* y $\mathbb{N}$, asociando a cada elemento de Σ^*, el número de elementos menores que el:

$$\begin{array}{ccc} \lambda & \longmapsto & 0 \\ a_1 & \longmapsto & 1 \\ a_2 & \longmapsto & 2 \\ & \cdots & \\ a_r & \longmapsto & r \\ a_1a_1 & \longmapsto & r+1 \\ a_1a_2 & \longmapsto & r+2 \\ & \cdots & \end{array}$$

El hecho de que esta aplicación sea una biyección acaba la demostración. ∎

Nótese que como consecuencia (Corolario) se tienen las propiedades siguientes:

[6] Un morfismo es una transformación $f : (G, *) \longrightarrow (T, \perp)$, que verifica $f(\lambda) = \lambda$ y $f(x * y) = f(x) \perp f(y)$, es decir, respeta el elemento neutro y la operación entre dos elementos del primer monoide se transforma en la operación entre las imágenes.

[7] Todos sus elementos conmutan al operarlos.

Corolario 5 *Sea Σ un alfabeto finito y $L \subseteq \Sigma^*$ un lenguaje. Entonces, L es un conjunto contable o, dicho en otras palabra, que a cada palabra se la puede asignar un número. Repetiremos esto a lo largo de los apuntes, pero esto es importante porque la forma de contar las palabras nos permitira decir cual es la primera palabra del lenguaje, que posición ocupa una palabra dada,etc..*
Más aún, el cardinal de los posibles lenguajes $L \subseteq \Sigma^*$ coincide con el número de subconjuntos de $\mathbb{N}$ y, por tanto, verifica:

$$\sharp(\mathcal{P}(\Sigma^*)) = \sharp(\mathcal{P}(\mathbb{N})) = \sharp(\mathbb{R}) = 2^{\aleph_0}.$$

En particular, hay una cantidad infinita no numerable de lenguajes sobre un alfabeto finito (cf. el ejemplo 25).

Operaciones básicas con palabras.

> ¿Por que tirabamos rocas contra los árboles? No se podia hacer otra cosa con las rocas
>
> Mel Brooks

Hasta ahora hemos visto varias propiedades de los lenguajes y las palabras, no hemos construido ningún lenguaje nosotros mismos. En esta sección veremos como, definiendo unas operaciones con las palabras, podremos definir nuevos e interesantes lenguajes que apareceran en diversas ocasiones a lo largo del curso. Además de la concatenación de palabras, podemos destacar las siguientes:

- *Potencia de Palabras:* Esta operación se define recursivamente a partir de la concatenación. Así, dada una palabra $x \in \Sigma^*$ y un número natural $n \in \mathbb{N}$, definimos la potencia x^n, mediante la siguiente función recursiva:
  - Definimos $x^0 = \lambda$,
  - para $n \geq 1$, definimos
    $$x^n := xx^{n-1}.$$
- *Reverso de una Palabra:* Al contrario que la operación anterior, esta operación no depende de ningún otro parametro. Es más, es una aplicación biyectiva
  $$^R : \Sigma^* \longrightarrow \Sigma^*,$$
  que se define utilizando esta recursión.
  - El reverso de la palabra vacia, es la palabra vacía $\lambda^R := \lambda$,
  - en cambio, si la palabra no es vacía entonces $x := x_1 y \in \Sigma^*$, donde $x_1 \in \Sigma$ y $y \in \Sigma^*$ y se tiene
    $$x^R := y^R x_1.$$

Un lenguaje que tendrá cierta relevancia en nuestras discusiones posteriores es el *Palíndromo* $\mathcal{P} \subseteq \{0,1\}^*$ y que viene dado por la siguiente igualdad:

$$\mathcal{P} := \{\omega \in \{0,1\}^* \mid \omega^R = \omega\}.$$

Aquí tenemos el primer ejemplo de lenguaje recursivamente enumerable. Fijemos por el momento el alfabeto $\Sigma := \{a, b\}$. Para mostrar que $\mathcal{P}$ es un lenguaje recursivamente enumerable necesitamos definir una función computable que dado una palabra nos devuelva 0 si la palabra no pertenece al lenguaje del palíndromo ó 1 en caso contrario.

La función f en nuestro caso, podría estar definida en Σ^* y devuelve

$$f(x) := \begin{cases} 1 & \text{si } x = x^R, \\ 0 & \text{en caso contrario.} \end{cases}$$

Esta función f no es única. Otra función que valdría para nuestros propositos esta dada por el Algorithmo 1. Este algoritmo también define una función computable.

Algorithm 1 Función característica de $\mathcal{P}$

Input: $x \in \Sigma^*$
Output: 1 si $x \in \mathcal{P}$, 0 en el caso contrario
  sea n la longitud de la palabra x
  for all $1 \leq i \leq n$ **do**
    if $x_i \neq x_{n-i}$ **then**
      return 0
    end if
  end for
  return 1

Además el algoritmo devuelve 0 o 1 para cualquier palabra, lo que significa que no entra en ningún bucle infinito. Notar que utilizando solamente la definición de reverso de una palabra no está claro que este algoritmo haga lo que nosotros queramos. Dejaremos la prueba de que el algoritmo devuelve 1 si y solamente si la palabra pertenece al lenguaje del palíndromo como ejercicio.

No hemos hablado de lenguajes no computables. Pero como primer ejemplo podriamos hablar de las cadenas de ADN que podría tener un ser vivo. Enunciado de esta forma, no esta claro que este lenguaje sea recursivamente enumerable, ya que no hay algoritmo conocido que permita saber, a partir de la expresión de una cadena de ADN si puede ser de un humano, un animal, un vegetal o un extraterreste.

Operaciones elementales con lenguajes

> Language forms a kind of wealth, which all can make use of at once without causing any diminution of the store, and which thus admits a complete community of enjoyment; for all, freely participating in the general treasure, unconsciously aid in its preservation.
>
> Auguste Comte

En la sección anterior hemos definido varias operaciones con palabras. Ahora, vamos a considerar las generalizaciones de las operaciones básicas con palabras aplicadas a los lenguajes formales. Para estas definiciones, suponemos definido un alfabeto Σ.

- *Unión de Lenguajes:* Dados $L_1, L_2 \subseteq \Sigma^*$, definimos la unión de dos lenguajes como:
$$L_1 \cup L_2 := \{x \in \Sigma^* \mid [x \in L_1] \vee [x \in L_2]\},$$
en otras palabras, la unión de lenguajes se realiza exactamente igual que la unión de conjuntos.

- *Concatenación de Lenguajes:* Dados $L_1, L_2 \subseteq \Sigma^*$, definimos la concatenación de dos lenguajes como:
$$L_1 \cdot L_2 := \{x \cdot y \in \Sigma^* \mid x \in L_1,\ y \in L_2\}.$$

- *Potencia de Lenguajes:* La potencia de un lenguaje se define recursivamente, como la potencia de una palabra. Dado un número entero positivo, n y un lenguaje L se define L^n como
$$L^n := \begin{cases} \{\lambda\} & \text{si } n = 0, \\ L \cdot (L^{n-1}) & \text{en otro caso.} \end{cases}$$

Nota 6 *Obsérvese que $L_1 \cdot L_2$ no es, en general, igual a $L_2 \cdot L_1$. Tampoco es cierto que si $L_1 \cdot L_2 = L_2 \cdot L_1$ entonces se tiene $L_2 = L_1$. El ejemplo más sencillo de esto es $\Sigma = \{a\},\ L_1 = \{a\},\ L_2 = \{aa\}$.*

Estas operaciones permiten definir lenguajes más complicados a partir de unos dados. Ahora podemos probar la siguiente relación entre la concatenación y la unión de varios lenguajes.

Proposición 7 (Distributivas) *Con las anteriores notaciones, se tienen las siguientes propiedades para lenguajes L_1, L_2 y L_3 contenidos en Σ^*:*

$$\begin{aligned} L_1 \cdot (L_2 \cup L_3) &= L_1 \cdot L_2 \cup L_1 \cdot L_3, \\ (L_1 \cup L_2) \cdot L_3 &= L_1 \cdot L_3 \cup L_2 \cdot L_3. \end{aligned}$$

Demostración. Otro ejercicio obvio. ∎

Otras operaciones importantes entre lenguajes hacen referencia al cálculo de la clausura transitiva por la operación de concatenación:

- *Clausura transitiva o monoide generado por un lenguaje:* Para un lenguaje $L \subseteq \Sigma^*$ definimos el monoide L^* como la siguiente unión infinita
$$L^* := \bigcup_{n \geq 0} L^n.$$
- *Clausura positiva de un lenguaje:* Dado un lenguaje $L \subseteq \Sigma^*$ definimos la clausura positiva L^+ que genera mediante la igualdad siguiente:
$$L^+ := \bigcup_{n \geq 1} L^n.$$

Nota 8 *Es obvio que L^* es la unión (disjunta si $\lambda \notin L$) entre L^+ y $\{\lambda\}$. En otro caso (i.e. si $\lambda \in L$), ambos lenguajes coinciden.*

1.3. Gramáticas formales

A. Thue[8] fue un matemático noruego que en 1914 introdujo la noción de sistema de reescritura. El interés de Thue era el análisis de los problemas de palabra para grupos y semi-grupos. Habrá que esperar a los trabajos de Noam Chomsky a finales de los años 50 para tener una clasificación de los lenguajes en el formato de gramáticas formales que, inicialmente, intentaba utilizar para modelizar solo los lenguajes naturales.

Definición 9 (Gramáticas Formales) *Una gramática formal esta dada por cuatro elementos (cuaterna) $G := (V, \Sigma, S, \mathcal{P})$, donde:*

- *V es un conjunto finito llamado alfabeto de símbolos no terminales o, simplemente, alfabeto de variables.*
- *Σ es otro conjunto finito, que verifica $V \cap \Sigma = \emptyset$ y se suele denominar alfabeto de símbolos terminales.*
- *$S \in V$ es una "variable" distinguida que se denomina variable inicial.*
- *$\mathcal{P} \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ es un conjunto finito llamado conjunto de producciones (o, simplemente, sistema de reescritura).*

La forma de definir una gramática solo tiene sentido despues de darle una utilidad. Seguidamente veremos la forma de operar sobre palabras usando una gramática, esto es, la dinámica asociada a una gramática. Para poder definir la dinámica asociada a una gramática, necesitamos definir como operar con los símbolos no terminales. Esto lo veremos en la siguiente definición.

[8] A. Thue. Probleme über Veränderungen von Zichereihen nach gegebenen reglen. *Regeln. Skr. Videnk. Selks. Kristiania I, Mat. Nat. Kl. :* **10** (1914).

Definición 10 *Sea $G := (V, \Sigma, S, \mathcal{P})$ una gramática, definiremos el conjunto de las formas sentenciales a*

$$(V \cup \Sigma)^* .$$

Dadas dos formas sentenciales s_1, s_2, decimos que $s_1 \to s_2$ si se verifica la siguiente propiedad:

$$\exists x, y, w, z \in (V \cup \Sigma)^* ,$$

tales que,

$$s_1 = x \cdot yw, \;\; s_2 = x \cdot zw, \quad (y, z) \in \mathcal{P},$$

ademas diremos que s_2 se deriva en un paso de s_1. También utilizaremos $s_1 \to^* s_2$ cuando s_2 se deriva en un número finito de pasos de s_1, es decir, si existen un número finito de formas sentenciales s_i, donde i esta en un rango de números naturales, tales que:

$$s_1 \to \ldots \to s_i \to s_{i+1} \to \ldots \to s_2.$$

Esta definición se aplicará a todas los tipos de gramáticas que utilizaremos, empezaremos dando un sencillo ejemplo.

Ejemplo 2 *Consideremos la gramática: $G := (V, \Sigma, S, \mathcal{P})$, donde*

$$V := \{S\}, \;\; \Sigma := \{a, b\}, \quad \mathcal{P} := \{(S, aS), (S, Sb), (S, \lambda)\}.$$

El conjunto de formas sentenciales esta formado por[9] *$\{S, a, b\}^*$ y un ejemplo de una computación sería:*

$$aaSbb \to aaaSbb \to aaaaSbb \to aaaa\lambda bb = aaaabb.$$

Nótese que las dos primeras veces hemos usado la producción (S, aS) y la última vez hemos usado (S, λ).

Notación 11 *Por analogía, se suele usar la notación $A \to B$ en lugar de $(A, B) \in \mathcal{P}$, para indicar una producción. Y, en el caso de tener más de una producción que comience en el mismo objeto, se suele usar $A \to B \mid C$, en lugar de escribir $A \to B$, $A \to C$.*

Notación BNF

> Una rosa con otro nombre no tendría mejor perfume
>
> "El nombre de la rosa"

La notación de Backus-Naur, también conocida como **BNF** (de Backus–Naur Form), es una notación alternativa para las gramáticas y que remonta su origen a la descripción que, del sánscrito, hizo el gramático hindú Panini. No es una notación estandarizada, aunque está bien establecida. Entre otras cosas porque los primeros usuarios de esta notación inisistieron en diversas notaciones para el símbolo $\to$. Aquí usaremos el "`estándar Wiki`" por llamarlo de algún modo. Se trata de hacer los siguientes cambios

[9] Por comodidad, quitaremos el subindice G.

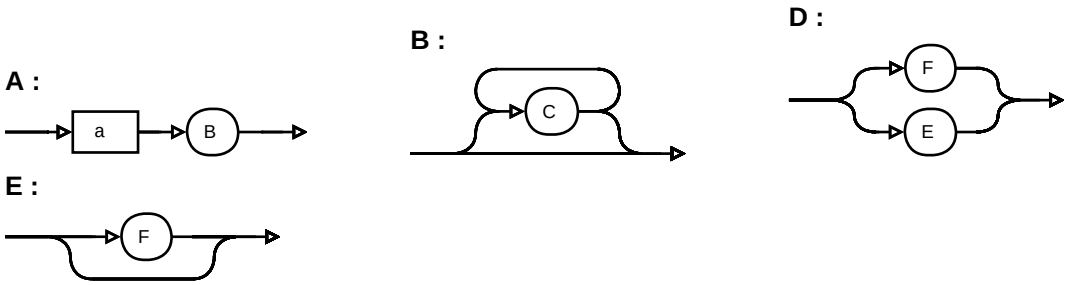


Figura 1.1: Representación EBNF A : 'a'B, $B : C^*$, $D : F \mid E$, $E : F?$

- Las variables $A \in V$ se representan mediante $\langle A \rangle \in V$.
- Los símbolos terminales (del alfabeto Σ) se escriben entre comillas (“a”, “b”, “c”, . . .)
- El símbolo asociado a las producciones $\to$ es reemplazado por ::=.

Así, la gramática descrita en el Ejemplo 2 anterior vendría dada por:

$$V := \{\langle S \rangle\}, \quad \Sigma := \{\text{“a”}, \text{“b”}\},$$

y las producciones estarían dadas por las reglas:

$$\langle S \rangle ::= \text{“a”}\langle S \rangle \;\mid\; \langle S \rangle\text{“b”} \;\mid\; \lambda.$$

Notación EBNF

> Standards fall in two categories: de facto and de jure
>
> Alfred Tannembaum

Esta notación es una extensión de la notación BNF. Es un estándar ISO-1497 y es utilizada (con algunas modificaciones) en los generadores de compiladores, como ANTLR. Se puede decir, en este caso, que es un estándar “de jure”, esto es, que esta regulado[10]. Avisamos al lector que bastantes generadores de compiladores implementan solamente la notación BNF, aunque esta notación es sumamente elegante y útil.

Básicamente, añade funcionalidad a la notación BNF, permitiendo repeticiones o diferentes opciones. Dejamos al lector varios ejemplos gráficos, (nótese la diferencia para los símbolos terminales y no terminales). Las siguientes son las principales modificaciones con respecto a la notación BNF,

- Las variables se denotan de la siguiente forma $A \in V$.
- Los símbolos terminales (del alfabeto Σ) se representan entre comillas simples.

[10] aunque no se utilice

- El símbolo asociado a las producciones $\rightarrow$ es reemplazado por :.
- Se introducen nuevos símbolos para representar repeticiones $*$ (ninguna, una o mas repeticiones), $+$ (una repetición al menos) y ? (una repetición a lo máximo).

Se deja como ejercicio al alumno hallar la expresión de la gramática,

$$\begin{aligned} \langle S \rangle &::= \text{“a”} \langle S \rangle, \\ \langle S \rangle &\qquad \lambda. \end{aligned}$$

utilizando la notación EBNF.
Independiente de las notaciones, el elemento clave es la noción de lenguaje generado por una gramática. En lo que respecta a este manuscrito, utilizaremos la notación usada en páginas anteriores (equivalente a BNF) y evitaremos (por complicada e innecesaria para nuestros propósitos) la notación EBNF.

Definición 12 (Lenguaje generado por una gramática) *Sea G una gramática definida como $G := (V, \Sigma, S, \mathcal{P})$. Llamaremos lenguaje generado por la gramática G al lenguaje $L(G) \subseteq \Sigma^*$ dado por:*

$$L(G) := \{x \in \Sigma^* \mid S \rightarrow^* x\},$$

es decir, a las palabras sobre el alfabeto de símbolos terminales alcanzables desde el símbolo inicial de la gramática.

1.4. Jerarquía de Chomsky

> Cualquiera que tenga forma puede ser definido, y cualquiera que pueda ser definido puede ser vencido
>
> Sun Tzu

Chomsky pretende la modelización de los lenguajes (formales y naturales) mediante gramáticas en su trabajo [Chomsky, 57]. El uso de máquinas con un número finito de estados (autómatas) ya aparece en [Chomsky–Miller, 57]. Al principio, estas abstracciones fueron pensadas para modelar la actividad cerebral, pero como muchas ideas, al final han tenido aplicaciones en muchas más areas.
Es en sus trabajos del año 59 (ca.[Chomsky, 59a] y [Chomsky, 59b]) donde aparece la clasificación que discutiremos en las páginas que siguen. En los siguientes capítulos veremos las relaciones que tienen las gramáticas con autómatas abstractos. Estos autómatas abstractos también tienen un papel importante en muchas aplicaciones de software.

Definición 13 (Gramáticas Regulares o de Tipo 3) *Definiremos las gramáticas con producciones lineales del modo siguiente:*

- *Llamaremos gramática lineal por la izquierda a toda* $G := (V, \Sigma, S, \mathcal{P})$ *gramática tal que todas las producciones de* $\mathcal{P}$ *son de uno de los dos tipos siguientes:*
  - $A \to a$, *donde* $A \in V$ *y* $a \in \Sigma \cup \{\lambda\}$.
  - $A \to aB$, *donde* $A, B \in V$ *y* $a \in \Sigma \cup \{\lambda\}$.
- *Llamaremos gramática lineal por la derecha a toda gramática* $G := (V, \Sigma, S, \mathcal{P})$ *tal que todas las producciones de* $\mathcal{P}$ *son de uno de los dos tipos siguientes:*
  - $A \to a$, *donde* $A \in V$ *y* $a \in \Sigma \cup \{\lambda\}$.
  - $A \to Ba$, *donde* $A, B \in V$ *y* $a \in \Sigma \cup \{\lambda\}$.
- *Llamaremos* **gramáticas regulares** *a las gramáticas lineales por la izquierda o lineales por la derecha.*

La dualidad (y simetría) entre las gramáticas lineales a izquierda o lineales a derecha es obvia y nos quedaremos solamente con las gramáticas lineales a izquierda.

Definición 14 (Lenguajes Regulares) *Un lenguaje* $L \subseteq \Sigma^*$ *se denomina un lenguaje regular si existe una gramática regular* $G := (V, \Sigma, S, \mathcal{P})$ *tal que* $L = L(G)$.

Por definición una producción puede ser una transformación del tipo $xAy \to w$, donde $x, y, w \in (\Sigma \cup V)^*$, $A \in V$. A las palabras α y β se las denomina *contexto* de la producción (o contexto de la variable A en esa producción). Así, una producción libre de contexto es una producción en la que ninguna variables tiene contexto, esto es, de la forma $A \to w$, con $A \in V$.

Definición 15 (Gramáticas libres de contexto o de Tipo 2) *Una gramática libre de contexto es una gramática* $G := (V, \Sigma, S, \mathcal{P})$ *tal que todas las producciones de* $\mathcal{P}$ *son del tipo siguiente:*

$$A \to w, \text{ donde } A \in V \text{ y } w \in (\Sigma \cup V)^*.$$

Un lenguaje **libre de contexto** *es un lenguaje generado por una gramática libre de contexto.*

El siguiente peldaño en la jerarquia de Chomsky son los lenguajes donde aparece el contexto. Pero con una gran restricción, las partes derechas de una producción tienen más longitud que la parte izquierda.

Definición 16 (Gramáticas sensibles al contexto o de Tipo 1) *Llamaremos gramática sensible al contexto a toda gramática* $G := (V, \Sigma, S, \mathcal{P})$ *tal que todas las producciones de* $\mathcal{P}$ *son del tipo siguiente:*

$$xAy \to xwy,$$

donde $A \in V$ *y* $x, y, w \in (\Sigma \cup V)^*$ *pero* $w \neq \lambda$. *Esto implica que en todas las producciones hay al menos una variable en la parte "izquierda" de la producción y la parte derecha tiene más longitud que la parte derecha.*
Un lenguaje **sensible al contexto** *es un lenguaje generado por una gramática sensible al contexto.*

Finalmente, llegamos a la forma más general que puede tener una gramática.

Definición 17 (Gramáticas formales de Tipo 0) *Llamaremos gramática formal de tipo 0 a toda gramática $G := (V, \Sigma, S, \mathcal{P})$ que admite todo tipo de producciones, esto es, sus producciones son de la forma*

$$x \to y,$$

donde $x, y \in (\Sigma \cup V)^*$, $x \neq \lambda$.

En las gramáticas de tipo 0 (las más generales) admitimos que haya producciones sin ninguna variable en el lado izquierdo de la producción.

1.5. Problemas y cuestiones

> La Ciencia es como el sexo:
> seguro que da alguna
> compensación práctica, pero no
> es por eso por lo que la hacemos
>
> Richard Feynman

Cuestiones relativas a lenguajes y gramáticas

Cuestión 1 *Se considera una gramática sobre el alfabeto $\Sigma := \{a, b\}$, cuyas producciones vienen dadas por*

$$S \to \lambda \mid aSa \mid bSb.$$

Decidir si el lenguaje generado por esa gramática es el conjunto de los palíndromos sobre Σ.

Cuestión 2 *Si el sistema de producciones de una gramática no posee ninguna transformación del tipo $A \to a$, ¿ podemos asegurar que no es una gramática regular?.*

Cuestión 3 *El lenguaje sobre el alfabeto $\{0,\ 1\}$ de las palabras que no contienen a 00 como subpalabra, ¿es un lenguaje regular?.*

Cuestión 4 *Dados dos lenguajes L_1 y L_2 sobre el alfabeto $\{a, b\}$, ¿ podemos asegurar que se verifica la siguiente igualdad*

$$(L_1 \cdot L_2)^R = L_1^R \cdot L_2^R?$$

Cuestión 5 *Dar una definición inductiva (recursiva) de la transformación $w \longmapsto w^R$ que revierte las palabras.*

Problemas relativos a lenguajes formales

Ejercicio 1 *Sea $\Sigma = \{0, 1\}$, encontrar la longitud de las siguientes palabras:*

- $x = 00111$,
- *x que es la representación binaria del número 9,*
- *x que es la palabra mas pequeña pertenenciente al siguiente lenguaje $L = \{x \mid$ x tiene 5 ceros más que 1$\}$.*

Ejercicio 2 *Clasificar según la jerarquía de Chomsky las siguientes gramáticas, donde $V = \{S, A\}$, $\Sigma = \{a, b\}$ y las producciones $\mathcal{P}$,*

- $\mathcal{P} = \{S \to aA, S \to a, S \to a\}$.
- $\mathcal{P} = \{S \to aA, S \to a, S \to Aa, A \to bb\}$.
- $\mathcal{P} = \{SS \to SaA, S \to Ab, A \to ba \mid AS\}$.
- $\mathcal{P} = \{SS \to \lambda, S \to Aaa, A \to ba \mid AS\}$.

Además, se pide encontrar una palabra de los lenguajes generados por estas gramáticas de longitud al menos 4 y utilizando todas las producciones que sean posibles.

Ejercicio 3 *Utilizar la notación EBNF con las gramáticas del ejercicio anterior.*

Ejercicio 4 *Sea $\Sigma = \{0, 1\}$, demostrar que el lenguaje de las representaciónes binarias de los números multiplos de ocho es un lenguaje regular.*

Ejercicio 5 *Sea $\Sigma = \{0, 1\}$, demostrar que el lenguaje de las representaciónes binarias de los numeros que tienen el mismo número de ceros que de unos es un lenguaje libre de contexto.*

Problemas relativos a gramáticas

Problema 1 *Sea $L := \{\lambda, a\}$ un lenguaje sobre el alfabeto $\Sigma := \{a, b\}$. Hallar L^n para los valores $n = 0, 1, 2, 3, 4$. ¿Cuántos elementos tiene L^n?*

Problema 2 *Dados los lenguajes $L_1 := \{a\}$ y $L_2 := \{b\}$ sobre el mismo alfabeto anterior, describir $(L_1 \cdot L_2)^*$ y $(L_1 \cdot L_2)^+$. Buscar coincidencias.*

Problema 3 *Probar que la concatenación de los lenguajes no es distributiva con respecto a la intersección de lenguajes.*

Problema 4 *Probar que la longitud $|\cdot| : \Sigma^* \longrightarrow \mathbb{N}$ es un morfismo de monoides suprayectivo, pero no es un isomorfismo excepto si $\sharp(\Sigma) = 1$.*

Problema 5 *Dado el alfabeto $\Sigma = \{0, 1\}$, se consideran los siguientes dos lenguajes:*

$$L_1 := \{\omega \in \Sigma^* \mid \sharp \text{ (ceros en } \omega) \in 2\mathbb{Z}\}.$$
$$L_2 := \{\omega \in \Sigma^* \mid \exists\, n \in \mathbb{N},\ \omega = 01^n\}.$$

Demostrar que $L_1 \cdot L_2$ es el lenguaje L_3 siguiente:

$$L_3 := \{\omega \in \Sigma^* \mid \sharp \text{ (ceros en } \omega) \in 2\mathbb{Z} + 1\}.$$

Problema 6 *Sea $G = (\{S\}, \{a, b\}, S, P)$ una gramática libre de contexto dada por las producciones:*

$$S \longrightarrow aSb \mid \lambda.$$

Probar que $L(G)$ es el lenguaje definido por

$$L := \{a^n b^n \mid n \in \mathbb{N}\}.$$

Problema 7 *Sea $L := \{a^n b^n c^n \mid n \in \mathbb{N}\}$ un lenguaje sobre el alfabeto $\Sigma = \{a, b, c\}$. Hallar una gramática G tal que $L(G) = L$.*

Problema 8 *Hallar una gramática libre de contexto (no regular) y otra equivalente regular para cada uno de los dos lenguajes siguientes:*

$$L_1 := \{ab^n a \mid n \in \mathbb{N}\},$$

$$L_2 := \{0^n 1 \mid n \in \mathbb{N}\}.$$

Problema 9 *Hallar gramáticas que generen los siguientes lenguajes:*

$$L_1 := \{0^m 1^n \mid [m, n \in \mathbb{N}] \wedge [m \geq n]\},$$

$$L_2 := \{0^k 1^m 2^n \mid [n, k, m \in \mathbb{N}] \wedge [n = k + m]\}.$$

Problema 10 *Dado el lenguaje $L := \{z \in \{a, b\}^* \mid \exists w \in \{a, b\}^*, \text{ con } z = ww\}$, hallar una gramática que lo genere.*

Problema 11 *Clasificar las siguientes gramáticas en términos de la jerarquía de Chomsky. Tratar de analizar los lenguajes generados por ellas y definirlos por compresión.*

a. $P := \{S \to \lambda \mid A, A \to c \mid AA\}$, $V := \{S, A\}$, $\Sigma := \{c\}$.

b. $P := \{S \to \lambda \mid A, A \to Ad \mid cA \mid c \mid d\}$, $V := \{S, A\}$, $\Sigma := \{c, d\}$.

c. $P := \{S \to c \mid ScS\}$, $V := \{S\}$, $\Sigma := \{c\}$.

d. $P := \{S \to c \mid AcA, A \to cc \mid cAc, cA \to cS\}$, $V := \{S, A\}$, $\Sigma := \{c\}$.

e. $P := \{S \to AcA, A \to 0, Ac \to AAcA \mid ABc \mid AcB, B \to B \mid AB\}$, $V := \{S, A, B\}$, $\Sigma := \{0, c\}$.

Problema 12 *Sea G la gramática dada por las siguientes producciones:*

$$S \to 0B \mid 1A,$$

$$A \to 0 \mid 0S \mid 1AA,$$
$$B \to 1 \mid 1S \mid 0BB.$$

Siendo $V := \{S, A, B\}$ y $\Sigma := \{0, 1\}$, probar que

$$L(G) := \{\omega \in \{0,1\}^* \mid \sharp\,(\text{ceros en } \omega) = \sharp\,(\text{unos en } \omega) \ \wedge \ |\omega| \geq 0\}.$$

Problema 13 *Probar que si L es el lenguaje dado por la siguiente igualdad:*

$$L := \{\omega \in \{0,1\}^* \mid \sharp\,(\text{ceros en } \omega) \neq \sharp\,(\text{unos en } \omega)\},$$

entonces $L^* = \{0,1\}^*$.

Problema 14 *Sea $L \subseteq \{a,b\}^*$ el lenguaje dado por la siguiente definición:*

- $\lambda \in L$,
- *Si $\omega \in L$, entonces $a\omega b \in L$ y $b\omega a \in L$,*
- *Si $x, y \in L$, entonces $xy \in L$.*

Describir el lenguaje y definirlo por comprensión.

Problema 15 *Probar que si L es generado por una gramática regular a izquierda, entonces L^R es generado por una gramática regular a derecha.*

Capítulo 2

Expresiones regulares

Índice

2.1. Introducción

> Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems
>
> Jamie Zawinski

Las expresiones regulares son formas compactas de generar un lenguaje de tipo 3 o lenguaje regular. Suelen estar relacionadas con el problema de buscar texto, realizar substituciones pero en realidad se pueden utilizar para resolver problemas más generales.

Generalmente, el problema que se pretende resolver mediante la introducción de las expresiones regulares es el de obtener algún tipo de descriptores para los lenguajes generados por las gramáticas regulares (las gramáticas de Tipo 3 o Regulares en la jerarquía de Chomsky), además de utilizarlos en la notación EBNF.

2.2. Las Nociones y algoritmos básicos

> A los niños les encantan las matemáticas, pero nos empeñamos en convencerles de lo contrario
>
> Francisco Santos

En las próximas subsecciones daremos una introducción a la semántica y gramática de las expresiones regulares.
Esta sección será de un corte bastante teórico pero pedimos al lector un poco de paciencia. Las preguntas que intentamos responder son suficientemente complicadas como para necesitar un tratamiento abstracto. Mientras intentabamos hacer el capítulo más ligero para el lector, el número de páginas crecía sin suavizar el "trauma".
Por esa razón y siendo éste un curso de lenguajes formales, utilizaremos la metodología propia del área y nos enfrentaremos sin miedo al formalismo. Empezaremos definiendo las reglas de formación (la gramática) de las expresiones regulares. A continuación las dotaremos de significado (la semántica) y veremos los recursos que nos ofrece esta nueva herramienta.

Definición 18 *Sea Σ un alfabeto finito. Llamaremos expresión regular sobre el alfabeto Σ a toda palabra sobre el alfabeto Σ_1 definido por la siguiente igualdad:*

$$\Sigma_1 := \{\emptyset, \lambda, +, \cdot, (,), ^*\} \cup \Sigma,$$

conforme a las reglas siguientes:

- *Las siguientes son expresiones regulares:*
  - *El símbolo $\emptyset$ es una expresión regular,*
  - *el símbolo λ es una expresión regular,*
  - *y el símbolo a es una expresión regular, para cualquier símbolo a en el alfabeto Σ,*
- *Si α y β son expresiones regulares, también lo son las construidas mediante las reglas siguientes:*
  - *$(\alpha + \beta)$ es una expresión regular,*
  - *$(\alpha \cdot \beta)$ es una expresión regular,*
  - *$(\alpha)^*$ es una expresión regular,*

Nota 19 *Por comodidad de la escritura (y sólo en el caso de que no haya ninguna posibilidad de ambigüedades) se suprimen los paréntesis y los símbolos de producto $(\cdot)$. También para evitar ambigüedades, el orden de prioridad será primero las expresiónes con parentesís, despues $\cdot$ y finalmente $+$.*

Nota 20 *La anterior definición no es sino la definición de un lenguaje sobre el alfabeto Σ_1: el lenguaje formado por las expresiones regulares. Dicha gramática se pude representar mediante una única variable $\langle S\rangle$, el alfabeto Σ_1 y las producciones siguientes:*

$$\begin{aligned}\langle S\rangle &\to \emptyset \mid \lambda \mid a, \quad a \in \Sigma,\\ \langle S\rangle &\to \langle S\rangle + \langle S\rangle \mid \langle S\rangle \cdot \langle S\rangle \mid (\langle S\rangle) \mid \langle S\rangle^*\end{aligned}$$

Ejemplo 3 *Tomemos el alfabeto $\Sigma := \{a, b\}$. Son expresiones regulares las secuencias de símbolos (palabras) siguientes:*

$$a \cdot a + b^* a, ab^* aa, \ldots$$

No serán expresiones regulares cosas del tipo:

$$(+a^*\emptyset)^*, \ldots$$

A cada objeto sintáctico, como lo es una expresión regular, conviene añadirle el mecanismo de asignación de significado (semántica). En el caso de expresiones regulares asignaremos *un único significado* a cada expresión como *el lenguaje formal que describe*.

Definición 21 *Sea Σ un alfabeto finito. A cada expresión regular sobre el alfabeto α le asignaremos un lenguaje formal $L(\alpha) \subseteq \Sigma^*$ conforme a las siguientes reglas:*

- *En el caso de que α no sea vacía, seguiremos las reglas siguientes:*
  - *Si $\alpha = \emptyset$, entonces $L(\alpha) = \emptyset$,*
  - *Si $\alpha = \lambda$, entonces $L(\alpha) = \{\lambda\}$,*
  - *Si $\alpha = a \in \Sigma$, entonces $L(\alpha) = \{a\}$,*
- *Aplicando las reglas recursivas, si α y β son dos expresiones regulares sobre el alfabeto Σ usaremos las reglas siguientes:*
  - *Si $\gamma = \alpha + \beta$, entonces $L(\gamma) = L(\alpha) \cup L(\beta)$,*
  - *si $\gamma = \alpha \cdot \beta$, entonces $L(\gamma) = L(\alpha) \cdot L(\beta)$,*
  - *Si $\gamma = \alpha^*$, entonces $L(\gamma) = L(\alpha)^*$,*

Ejemplo 4 *A modo de ejemplo, sea $\alpha := a^*ba^*$ la expresión regular sobre el alfabeto $\Sigma := \{a, b\}$. Entonces, el lenguaje generado por α es*

$$L(a^*ba^*) = L(a)^* \cdot L(b) \cdot L(a)^* = \{a^m ba^n, n, m \in \mathbb{N}\}.$$

Definición 22 *Diremos que dos expresiones regulares α y β son tautológicamente equivalentes (o, simplemente, equivalentes) si se verifica:*

$$L(\alpha) = L(\beta)$$

Escribiremos $\alpha \equiv \beta$ para indicar equivalencia tautológica.

Algunas de las propiedades básicas de la asignación semántica de lenguajes a expresiones regulares se resumen en la siguiente proposición, cuya demostración es completamente obvia.

Proposición 23 (Propiedades básicas) *Sea Σ un alfabeto finito, se verifican las siguientes propiedades para expresiones regulares α, β, γ sobre Σ:*

a. Asociativas,

$$\alpha \cdot (\beta \cdot \gamma) \equiv (\alpha \cdot \beta) \cdot \gamma, \quad \alpha + (\beta + \gamma) \equiv (\alpha + \beta) + \gamma.$$

b. Conmutativa (sólo para $+$) [1] *:*

$$\alpha + \beta \equiv \beta + \alpha.$$

c. Elementos neutros:

$$\alpha + \emptyset \equiv \alpha, \quad \alpha \cdot \lambda \equiv \alpha, \quad \alpha \cdot \emptyset \equiv \emptyset.$$

d. Idempotencia:

$$\alpha + \alpha \equiv \alpha.$$

e. Distributivas:

$$\alpha \cdot (\beta + \gamma) \equiv \alpha \cdot \beta + \alpha \cdot \gamma.$$

$$(\alpha + \beta) \cdot \gamma \equiv \alpha \cdot \gamma + \beta \cdot \gamma.$$

f. Invariantes para *:

$$\lambda^* \equiv \lambda, \quad \emptyset^* \equiv \emptyset.$$

g. La notación α^+:

$$\alpha^* \cdot \alpha \equiv \alpha \cdot \alpha^* \equiv \alpha^+.$$

h. $\alpha^* = \lambda + \alpha^+$

i. Relación de * con la suma:

$$(\alpha + \beta)^* \equiv (\alpha^* \beta^*)^*.$$

Dejamos la prueba de esta proposición como ejercicio.

[1] Aunque la insistencia sea innecesaria, es común olvidar que $2 \cdot 3$ no es igual que $3 \cdot 2$. Cosas de malos hábitos.

2.3. De RE's a RG's: el método de las derivaciones

> Algorithms are the
> computational content of proofs.
>
> Robert Harper

En esta sección demostraremos que todo lenguaje generado por una expresión regular se puede generar también por una gramática regular[2]. Esto será un primer paso para mostrar que los lenguajes regulares son sólo aquellos que están generados por una expresión regular. Las pruebas de esta sección están diseñadas para ser traducidas a algoritmos. Queremos remarcar que los algoritmos en esta sección están pensados para ser facilmente implementables pero no están optimizados.
Antes de mostrar el algoritmo que toma una expresión regular y devuelve una gramática regular que genera el mismo lenguaje, dedicaremos algún tiempo a fijar una de las operaciones básicas en el tratamiento de expresiones regulares: la derivación.

Definición 24 *Sea Σ un alfabeto finito, $a \in \Sigma$ un símbolo del alfabeto, y α una expresión regular sobre el alfabeto Σ. Llamaremos derivada de α con respecto al símbolo α a la expresión regular $\frac{\partial \alpha}{\partial a}$ definida mediante la regla recursiva siguiente:*

- *Para expresiones regulares de longitud 1, tenemos las definiciones siguientes:*

$$\frac{\partial \emptyset}{\partial a} = \emptyset, \quad \frac{\partial \lambda}{\partial a} = \emptyset, \quad \frac{\partial a}{\partial a} = \lambda, \quad \frac{\partial b}{\partial a} = \emptyset, \quad \forall b \in \Sigma,\ b \neq a.$$

- *Si α y β son dos expresiones regulares sobre Σ, definiremos:*

$$\frac{\partial(\alpha)^*}{\partial a} = \frac{\partial(\alpha)}{\partial a} \cdot \alpha^*,$$
$$\frac{\partial(\alpha+\beta)}{\partial a} = \frac{\partial \alpha}{\partial a} + \frac{\partial \alpha}{\partial a},$$
$$\frac{\partial(\alpha \cdot \beta)}{\partial a} = \frac{\partial \alpha}{\partial a}\beta + t(\alpha)\frac{\partial \beta}{\partial a},$$

donde $t(\alpha)$ es la función dada por la identidad siguiente:

$$t(\alpha) := \begin{cases} \lambda & \text{si } \lambda \in L(\alpha), \\ \emptyset & \text{en caso contrario.} \end{cases} \tag{2.1}$$

Nota 25 *La derivada de una expresión regular con respecto a un símbolo de un alfabeto finito es, claramente, una derivada parcial y, por tanto, está perfectamente descrita mediante el símbolo $\frac{\partial \alpha}{\partial a}$. Sin embargo, el símbolo ∂ parece poner nerviosos a ciertos autores, por lo que también es costumbre (solamente entre los nerviosos) usar el símbolo menos correcto (pero menos enervante) $D_a(\alpha)$. Dejaremos que los alumnos reescriban la definición anterior con esta nueva notación. De ahora en adelante usaremos $D_a(\alpha)$.*

[2] aunque deberiamos especificar, las ideas que damos se pueden aplicar para dar una gramática regular por la izquierda o por la derecha.

La propiedad fundamental por la cual derivar es una acción útil, viene dada por la siguiente proposición (cuya prueba omitiremos por obvia).

Proposición 26 *Con las notaciones anteriores, para cada expresión regular α sobre un alfabeto Σ, la derivada $D_a(\alpha)$ es una expresión regular que verifica la siguiente propiedad:*

$$L(D_a(\alpha)) := \{x \in \Sigma^* \mid \quad ax \in L(\alpha)\}.$$

Demostración. Como pista para la demostración, digamos que sale de manera inmediata a partir de la definición recursiva de expresiones regulares. ∎

Una identificación más clara de la relación de una palabra con sus derivadas viene dada por la siguiente proposición (que resume la regla de Leibnitz para polinomios homogéneos multivariados).

Proposición 27 (Regla de Leibnitz para Expresiones Regulares) *Dada una expresión regular α sobre un alfabeto finito Σ, supongamos que $\Sigma = \{a_1, \ldots, a_r\}$. Se tiene la siguiente equivalencia,*

$$\alpha \equiv a_1 D_{a_1}(\alpha) + a_2 D_{a_2}(\alpha) + \cdots + a_r D_{a_r}(\alpha) + t(\alpha),$$

donde $t(\alpha)$ es la función definida en la ecuación 2.1.

Demostración. Mediante la proposición anterior, basta con verificar a que las palabras en $L(\alpha)$ son de los tipos (obvios) siguientes: o empiezan por algún símbolo de $a \in \Sigma$ (y, por tanto, están en $L(D_a(\alpha))$) o es la palabra vacía (y queda sumida en la expresión $t(\alpha)$). El caso restante es que no haya ninguna palabra en $L(\alpha)$ lo cual también queda expresado por la identidad y por $t(\alpha)$. ∎

La derivación de una expresión regular juega un papel importante para demostrar que todo lenguaje descrito por una expresión regular es un lenguaje regular, es decir, que existe una gramática regular que lo genera. En unas lineas, describiremos un algoritmo que transforma expresiones regulares en gramáticas regulares respetando los lenguajes que describen/generan: es el llamado *Método de las Derivaciones*.

Lema 28 *Sea L_1 y L_2 dos lenguajes (regulares) sobre el alfabeto Σ generados respectivamente por gramáticas $G_1 := (V', \Sigma, S_1, \mathcal{P}')$ y $G_2 := (V'', \Sigma, S_2, \mathcal{P}'')$, entonces $L_1 \cup L_2$ es también un lenguaje (regular) generado por una gramática. La gramática que genera la unión es una nueva gramática $G = (V, \Sigma, S, \mathcal{P})$ dada por las reglas siguientes:*

a. *Al precio de renombrar las variables, podemos suponer que $V' \cap V'' = \emptyset$ (es decir, G_1, G_2 no poseen símbolos no terminales comunes) y $\mathcal{P}' \cap \mathcal{P}'' = \emptyset$.*

b. *Introducimos una nueva variable $S \notin V' \cup V''$.*

c. *Finalmente, definimos $V := V' \cup V'' \cup \{S\}$ y definimos $\mathcal{P} := \{S \to S_1 \mid S_2\} \cup \mathcal{P}'' \cup \mathcal{P}'$.*

Demostración. Con esta definición de la nueva gramática G es un mero ejercicio de demostración por inducción en el número de pasos de cálculo. ∎

En el caso de unión finita $L = L_1 \cup L_2 \cup \cdots$, el lema anterior se puede extender de la forma obvia. La unión (finita) de diferentes lenguajes se puede generar con una gramática a lo sumo tan "complicada". Enunciaremos un pequeño resultado para lenguajes regulares sin demostrarlo que expresa de forma formal exactamente eso.

Lema 29 *La unión finita de lenguajes generados por gramáticas regulares es un lenguaje generado por una gramática regular.*

Lema 30 *Sea $L \subseteq \Sigma^*$ un lenguaje sobre el alfabeto Σ generado por una gramática (regular) $G = (V, \Sigma, S, \mathcal{P})$. Sea $a \in \Sigma$ un símbolo del alfabeto. Entonces, la siguiente gramática $G_1 = (V', \Sigma, S_1, \mathcal{P}')$ genera $a \cdot L$:*

- *Sea S_1 una nueva variable (no presente en V) y definamos $V' := V \cup \{S_1\}$.*
- *Definamos $\mathcal{P}' := \mathcal{P} \cup \{S_1 \to aS\}$.*

Demostración. De nuevo un mero ejercicio de demostración por inducción. Es importante señalar que si la gramática G es regular, la nueva gramática también es regular. ∎

Combinando la Proposición 27 con los Lemas 29 y 30, uno pensaría en un argumento inductivo para generar un lenguaje dado por una expresión regular α a partir de sus derivadas. La idea, grosso modo, sería la siguiente:

Sea $L(\alpha)$ un lenguaje dado por una expresión regular α sobre un alfabeto Σ, supongamos que $\Sigma = \{a_1, a_2, \ldots, a_r\}$. Entonces, la Regla de Leibnitz para expresiones regulares nos da la siguiente identidad:

$$L(\alpha) = a_1 \cdot L(D_{a_1}(\alpha)) \cup a_2 \cdot L(D_{a_2}(\alpha)) \cup \cdots \cup a_r \cdot L(D_{a_r}(\alpha)) \cup t(\alpha).$$

A partir de esta identidad, uno pretende generar un árbol entre expresiones regulares y podría tratar de argumentar como sigue:

- Supongamos dadas gramáticas $G_1, G_2, \ldots, G_r$ que generan (respectivamente) los lenguajes $L(D_{a_1}(\alpha)), L(D_{a_2}(\alpha)), \ldots, L(D_{a_r}(\alpha))$.
- Utilizando el Lema 30, uno podría construir gramáticas $G_1', G_2', \ldots, G_r'$ de tal modo que generan los lenguajes $a_1 L(D_{a_1}(\alpha)), a_2 L(D_{a_2}(\alpha)), \ldots, a_r L(D_{a_r}(\alpha))$.
- Finalmente, utilizando el Lema 29 uno concluiría exhibiendo la gramática que genera el lenguaje $L(\alpha)$ a través de la identidad dada por la Regla de Leibnitz (Proposición 27).

El problema en esta forma de pensamiento es la "gradación" de las gramáticas. En esta propuesta hay implícitamente una suposición de que las expresiones regulares asociadas a las derivadas son "más pequeñas" que la expresión regular original. El concepto de "más pequeño" es inevitable para poder dar un argumento recursivo con

esta construcción. Sin embargo, la intuición sobre las propiedades de las derivadas no debe confundirnos. La derivada de una expresión regular puede ser "más grande" (o de mayor "grado") que la expresión original, debido justamente al papel del operador *. Veamos algunos ejemplos:

Ejemplo 5 *Sea $\Sigma = \{a, b\}$ y consideremos la expresión regular $a^* \subseteq \Sigma^*$. Consideramos las derivadas $D_a(a^*) = a^*$, $D_b(a^*) = \emptyset$. Tendremos, por Leibnitz,*

$$\{a\}^* = L(a^*) = a \cdot L(a^*) + \emptyset + \{\lambda\}.$$

Claramente, la inducción pretendida nos dice que para hallar la gramática asociada a la expresión a^* **¡necesitamos calcular previamente la gramática asociada a la expresión a^*!**. *La respuesta a este dilema en este caso, sería la gramática siguiente:*

- *Dado que $\lambda \in L(a^*)$ escribamos la producción $S \to \lambda$,*
- *Dado que $D_a(a^*) \neq \lambda, \emptyset$, escribamos la producción $S \to aS$.*

Nótese que, en este ejemplo, hemos identificado la variable S con la expresión regular a^* y, hemos escrito la producción $S \to aS$ porque $D_a(a^*) = a^*$.

Ejemplo 6 *En el anterior ejemplo, la expresión regular obtenida tras derivar no "crece" con respecto a la expresión regular original (en todo caso, se estabiliza). Pero es posible que se produzca un crecimiento (al menos en la longitud como palabra) y eso se muestra a través del ejemplo $(abc)^*$ de una expresión regular sobre el alfabeto $\Sigma = \{a, b, c\}$. Al derivar observamos:*

$$D_a((abc)^*) = bc(abc)^*,$$

cuya longitud es mayor que la longitud de la expresión regular original.

Para resolver este problema acudiremos al análisis de las derivadas sucesivas de una expresión regular.

Definición 31 (Derivadas sucesivas (de una RE)) *Sea*

$$\Sigma = \{a_1, a_2, \ldots, a_r\},$$

un alfabeto finito, $x \in \Sigma^*$ una palabra sobre el alfabeto y α una expresión regular. Definiremos la derivada $D_x(\alpha)$ mediante el proceso siguiente:

- *Si $x = \lambda$ es la palabra vacía, $D_\lambda(\alpha) = \alpha$.*
- *Si la longitud de la palabra x es uno y, por tanto, $x = a \in \Sigma$, definimos $D_x(\alpha) = D_a(\alpha)$, conforme a la definición de derivada anterior.*
- *Si x es una palabra de longitud n mayor que 2 y, por tanto, existe $a \in \Sigma$ y existe $y \in \Sigma^*$, tal que $x = ay$ definimos*

  $$D_x(\alpha) = D_y(D_a(\alpha)),$$

  conforme a la definición recursiva para palabras de longitud $n - 1$.

Nota 32 *De nuevo la intuición puede hacer estragos, nótese que no hay conmutatividad de las derivadas (como sí ocurría si estuvieramos en el caso de las derivadas parciales habituales). En general, dada una expresión regular α, se tiene que $D_{ab}(\alpha) \neq D_{ba}(\alpha)$. Por poner un ejemplo, consideremos la expresión $\alpha = aa^*bb^*$. Se tiene ,*

$$D_a(\alpha) = a^*bb^*, \quad D_b(\alpha) = \emptyset.$$

Por tanto,

$$D_{ba}(\alpha) = D_a(D_b(\alpha)) = D_a(\emptyset) = \emptyset,$$

mientras que

$$D_{ab}(\alpha) = D_b(D_a(\alpha)) = D_b(a^*bb^*) = b^*.$$

Como una nota para el lector, comentemos que la definición de $D_x(\alpha)$ esta dada de forma que la expresión regular resultante define un lenguaje muy especial, que se puede dar de una forma parecida a la regla de Leibnitz. Se deja este comentario para que el lector reflexione y pasaremos al siguiente resultado del curso.

Proposición 33 *Sea α una expresión regular sobre un alfabeto finito Σ y sea $Der(\alpha)$ el conjunto de todas las derivadas sucesivas de α con respecto a palabras en Σ^*. Esto es,*

$$Der(\alpha) := \{\beta \mid \quad \exists x \in \Sigma^*,\ D_x(\alpha) = \beta\}.$$

Entonces, $Der(\alpha)$ es un conjunto finito.

Demostración. Se demostraría por inducción en la definición recursiva de la expresión regular. ∎

Nuestro propósito es construir un grafo con pesos asociado al conjunto de todas las derivadas de la expresión regular. Esto va a constituir la gramática buscada.

Proposición 34 *El algoritmo 2 transforma toda expresión regular α en una gramática finita G que genera el lenguaje $L(\alpha)$ descrito por la expresión. En particular, los lenguajes descritos por expresiones regulares son lenguajes regulares.*

Demostración. La idea principal para realizar este algoritmo es la Regla de Leibnitz, combinando las gramáticas con los Lemas 29 y 30. ∎

2.4. De RG's a RE's: uso del lema de Arden

> System is what differentiates the professional from the amateur.
>
> John T. Barlett

En esta sección veremos el algoritmo inverso del propuesto en la sección anterior. En otras palabras, a partir de una gramática regular, daremos una expresión regular que genera el mismo lenguaje que la gramática regular. Para ello, introduciremos ecuaciones lineales relacionados con el lema de Arden.

Algorithm 2 Algoritmo de RE a RG

Input: Una expresión regular α sobre un alfabeto finito Σ

Output: Una gramática G que genera $L(\alpha)$

Hallar $Der(\alpha) := \{D_x(\alpha) \mid x \in \Sigma^*\}$
Definir un conjunto finito V de variables con el mismo número de elementos que $Der(\alpha)$. Sea $S \in V$ un elemento de ese conjunto de variables.
Definir una aplicación E del conjunto de las derivaciones $Der(\alpha)$ al conjunto de variables V, tal que $E(\alpha) = S$.
$\mathcal{P} := \emptyset$
if $\lambda \in L(\alpha)$ **then**
  $\mathcal{P} := \{S \to \lambda\}$
end if
$nuevaP := True$
while nuevaP **do**
  $nuevaP := False$
  for all $\beta \in Der(\alpha)$ **do**
    for all $a \in \Sigma$ **do**
      $\gamma := D_a(\beta)$
      $B, C := E(\beta), E(\gamma)$
      if $\lambda \in \beta$ and $\{B \to \lambda\} \notin \mathcal{P}$ **then**
        $\mathcal{P} := \mathcal{P} \cup \{B \to \lambda\}$
        $nuevaP := True$
      end if
      if $\{B \to aC\} \notin \mathcal{P}$ **then**
        $\mathcal{P} := \mathcal{P} \cup \{B \to aC\}$
      end if
    end for
  end for
end while
return $(V, \Sigma, S, \mathcal{P})$

Ecuaciones lineales

La ecuaciones lineales en los lenguajes regulares juegan un papel muy importante. Estás nos posibilitarán probar que las palabras generadas por una gramática regular forman un lenguaje dado por una expresión regular. Empecemos con la definición.

Definición 35 *Llamaremos sistema de ecuaciones lineales en expresiones regulares a toda ecuación del tipo siguiente:*

$$\begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} = \begin{pmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{n,1} & \cdots & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} + \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \tag{2.2}$$

donde los $\alpha_{i,j}$ y los β_k son expresiones regulares sobre un alfabeto Σ.

Una solución de uno de tales sistemas de ecuaciones es una lista $(\omega_1, \ldots, \omega_n)$ de expresiones regulares sobre el mismo alfabeto, tales que

$$\omega_i \equiv \alpha_{i,1} \cdot \omega_1 + \cdots + \alpha_{i,n} \cdot \omega_n + \beta_i,$$

donde $\equiv$ es la igualdad entre los lenguajes que describen (i.e. la igualdad tautológica de las expresiones regulares).
El objetivo de esta subsección es la discusión del método obvio de resolución de este tipo de ecuaciones lineales. La clave para poder establecer lo obvio es un clásico resultado de Arden:

Definición 36 *Se denomina ecuación lineal fundamental en expresiones regulares a la ecuación lineal en una variable X siguiente:*

$$X \equiv \alpha X + \beta$$

donde α y β son expresiones regulares sobre un alfabeto finito Σ.

La ecuación lineal fundamental es el sistema de ecuaciones lineales en expresiones regulares más sencillo que se puede definir. Por ello, vamos a empezar estudiando este caso por separado.

Lema 37 (Lema de Arden) *Dada la ecuación fundamental siguiente:*

$$X \equiv \alpha X + \beta,$$

donde α, β son expresiones regulares sobre un alfabeto Σ. Se verifican las propiedades siguientes:

a. La ecuación fundamental anterior posee una solución única si y solamente si $\lambda \notin L(\alpha)$.

b. La expresión regular $\alpha^* \cdot \beta$ es siempre solución de la ecuación fundamental anterior.

c. Si $\lambda \in L(\alpha)$ para cualquier expresión regular γ, la expresión $\alpha^*(\beta + \gamma)$ es una solución de la ecuación fundamental.

Demostración. Aunque no se pretende dar una demostración completa del lema, al menos señalaremos los hechos fundamentales.
Se puede ver fácilmente que cualquier expresión regular que sea solución debe contener al lenguaje $L(\alpha)^* \cdot L(\beta)$. Otra observación trivial es que cualquier palabra del lenguaje generado por una solución debe de estar en el lenguaje generado por β o es la concatenación de dos palabras, la primera en el lenguaje generado por α y la segunda en el lenguaje generado por X.
También nótese que si α es una expresión regular, se tiene que

$$L(\alpha) \cdot L(\alpha)^* \equiv L(\alpha)^+.$$

Es decir, $\alpha \cdot \alpha^* \equiv \alpha^+$. Ahora bien, nótese que $L(\alpha)^* = L(\alpha)^+$ si y solamente si $\lambda \in L(\alpha)$.
Del mismo modo, consideremos una expresión regular γ cualquiera, tendremos:

$$\alpha \cdot \alpha^* \cdot (\beta + \gamma) + \beta \equiv \alpha^+ \cdot \beta + \beta + \alpha^+ \cdot \gamma \equiv (\alpha^+ + \lambda) \cdot \beta + \alpha^+ \cdot \gamma \equiv \alpha^* \cdot \beta + \alpha^+ \cdot \gamma,$$

y también tenemos $\alpha^* \cdot (\beta + \gamma) = \alpha^* \cdot \beta + \alpha^* \cdot \gamma$. Esto nos da inmediatamente que si $\alpha^* \equiv \alpha^+$ o si $\gamma = \emptyset$ tenemos la equivalencia. ∎

Este simple lema es la base para el algoritmo obvio de sustitución, es decir, eligiendo una variable y sustituyéndola en las demás ecuaciones. Formalmente, esto se expresa de la siguiente manera.

Proposición 38 *Toda ecuación lineal en expresiones regulares del tipo de la Definición 35 posee solución, que es una lista de expresiones regulares sobre el mismo alfabeto.*

Demostración. El algoritmo se divide en las dos fases obvias: triangulación/reducción (a través del lema de Arden) y levantamiento (invirtiendo las expresiones ya despejadas). Los detalles del algoritmo se dejan como ejercicio al alumno.

- **Triangulación:** Seguiremos la notación dada en la ecuación 2.2 y procederemos utilizando inducción. El caso de solo una ecuación se resuelve mediante el Lema de Arden. Para el caso $n > 1$, usaremos un doble paso:
  - *Despejar.* Podemos despejar X_n en la última ecuación, mediante la expresión siguiente:
    $$X_n := \alpha_{n,n}^* R_n, \tag{2.3}$$
    donde $R_n := \sum_{j=1}^{n-1} \alpha_{n,j} \ + \beta_n$.
  - *Sustituir.* Podemos sustituir la expresión anterior en el resto de las ecuaciones obteniendo un nuevo sistema de $(n-1)$ ecuaciones en $(n-1)$ variables. Este sistema viene dado, obviamente, por las expresiones siguientes para $1 \leq i \leq n-1$:
    $$X_i := \left(\sum_{j=1}^{n-1} \left(\alpha_{i,j} + \alpha_{i,n} \alpha_{n,n}^* \alpha_{n,j} \right) X_j \right) + \left(\beta_i + \alpha_{n,n}^* \beta_n \right).$$

- **Levantamiento.** Una vez llegados al caso $n = 1$, se obtiene una expresión regular válida para X_1 y se procede a levantar el resto de las variables usando las expresiones obtenidas en la fase de despejado (expresiones 2.3).

Notar que cuando se encuentra la solución para X_1 esta no depende de las variables $X_1, \ldots, X_n$. De esta forma, queda probada la proposición. ∎

Un buen ejercicio para el lector puede ser investigar la complejidad de un algoritmo definido de esta manera. Otra observación es que el algoritmo empieza sutituyendo la última variable en las demás ecuaciones.
Esta elección es totalmente arbitraria, cualquier elección para sustituir la variable da una expresión regular que genera el mismo lenguaje aunque esto no significa que es el mismo resultado.
Ahora que podemos resolver sistemas lineales en expresiones lineales, demostraremos como la solución de un sistema lineal da una expresión regular que resolverá nuestro problema. Comenzaremos asociando a cada gramática regular $G := (V, \Sigma, S, \mathcal{P})$ un sistema de ecuaciones lineales en expresiones regulares mediante la regla siguiente:

- Supongamos $V := \{S, X_1 \ldots, X_n\}$ es el conjunto de los símbolos no terminales, que supondremos de cardinal $n + 1$. Definamos un conjunto de variables $\{Y_0, Y_1 \ldots, Y_n\}$ con el mismo cardinal y con la asignación como biyección.

- Definamos para cada i, $0 \leq i \leq n$, la expresión regular β_i mediante la construcción siguiente. Consideremos todas las producciones que comienzan en la variable $A \in V$ y terminan en elementos de $\Sigma \cup \{\lambda\}$. Supongamos que para esa variable $A := X_i$ tengamos

  $$A \to a_1 \mid \ldots$$

  Definimos [3]

  $$\beta_i := a_1 + \ldots$$

  Si, para está variable, no hay de este tipo de producciones, definiremos $\beta_i := \emptyset$.

- Para cada i y para cada j, definiremos el coeficiente $\alpha_{i,j}$ del modo siguiente. Consideremos todas las producciones que comienzan en el símbolo no terminal $B := X_i$ e involucran al símbolo no terminal $C := X_j$. Supongamos que tales producciones sean:

  $$B \to (a_1 \mid \cdots)C,$$

  entonces definiremos

  $$\alpha_{i,j} := a_1 + \cdots$$

  Igual que anteriormente, si no hubiera ninguna de tales producciones, definiremos $\alpha_{i,j} := \emptyset$.

[3] Note el lector que incluimos las producciónes que en la parte derecha tenga λ.

Definición 39 (Sistema asociado a una gramática) *Dada una gramática $G = (V, \Sigma, S, \mathcal{P})$ llamaremos sistema asociado a G y lo denotaremos por $\mathcal{S}(G)$ al sistema:*

$$\mathcal{S}(G) := \left\{ \begin{pmatrix} Y_0 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} & \cdots & \alpha_{0,n} \\ \vdots & \ddots & \vdots \\ \alpha_{n,0} & \cdots & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} Y_0 \\ \vdots \\ Y_n \end{pmatrix} + \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_n \end{pmatrix} \right\},$$

dado por las anteriores reglas de construcción.

Proposición 40 *Con las anteriores notaciones, sea $(\omega_0, \ldots, \omega_n)$ una solución del sistema $S(G)$ asociado a una gramática G. Entonces, $L(G)$ es el lenguaje generado por la expresión regular ω_0.*

Demostración. La idea de la demostración es que estamos asociando una expresión regular a cada variable. La variable Y_i es la expresión regular de las palabras que se pueden generar a través de derivaciones empezando por la variable X_i si $i > 0$. Por esa razón la solución de nuestro problema es encontrar Y_0. A partir de esta idea, la demostración se realiza por inducción. ∎

Como una pequeña nota, en la Proposición 38, solamente es necesario hallar una de las variables. Es decir, no es necesario hallar la solución completa.

Teorema 41 *Los lenguajes regulares son los descritos por las expresiones regulares. Es decir, todo lenguaje descrito por una expresión regular es el lenguaje generado por alguna gramática regular y, recíprocamente, todo lenguaje generado por alguna gramática regular puede ser descrito por alguna expresión regular. Además, existen algoritmos que transforman RE's en RG's. El recíproco también es cierto.*

Demostración. Basta con combinar los algoritmos descritos en las Proposiciones 40 y 34. ∎

2.5. Problemas y cuestiones

> La Ciencia es como el sexo: seguro que da alguna compensación práctica, pero no es por eso por lo que la hacemos
>
> Richard Feynman

Cuestiones relativas a expresiones regulares

Cuestión 6 *Se dice que una expresión regular α está en forma normal disyuntiva si*

$$\alpha = \alpha_1 + \cdots + \alpha_n,$$

donde las expresiones regulares $\alpha_1, \ldots, \alpha_n$ *no involucran el operador* $+$*. Decidir si la siguiente expresión regular esta en forma disyuntiva o encontrar una forma de ponerla en forma disyuntiva:*

$$(0 + 00 + 10)^*,$$

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

Cuestión 7 *Decidir si es verdadera la siguiente igualdad de expresiones regulares:*

$$(a + b)^* = (a^* + b^*)^*.$$

Cuestión 8 *¿Pertenece la palabra acdcdb al lenguaje descrito por la expresión regular siguiente:*

$$\alpha = (b^*a^* (cd)^* b)^* + (cd)^*?$$

Cuestión 9 *Sea* L *el lenguaje sobre el alfabeto* $\{a, b\}$ *formado por todas las palabras que contienen al menos una aparición de la palabra* b*. ¿Es* L *el lenguaje descrito por la expresión regular siguiente*

$$\alpha := a^* (ba^*)^* bb^* (b^*a^*)^*?$$

Cuestión 10 *Dada cualquier expresión regular* α*, ¿Se cumple* $\alpha^*\alpha = \alpha^*$*?*

Cuestión 11 *Dadas tres expresiones regulares* α, β, γ*, ¿Es cierto que* $\alpha + (\beta \cdot \gamma) \equiv (\alpha + \beta) \cdot (\alpha + \gamma)$*?*

Cuestión 12 *¿Es siempre la derivada de una expresión regular otra expresión regular?*

Problemas relativos a expresiones regulares

Problema 16 *Dadas* α, β *dos expresiones regulares. Probar que si* $L(\alpha) \subseteq L(\beta)$*, entonces* $\alpha + \beta \equiv \beta$.

Problema 17 *Dada la expresión regular* $\alpha = a + bc + b^3a$*, ¿Cuál es el lenguaje regular* $L(\alpha)$ *descrito por ella?. ¿Cuál es la expresión regular que define el lenguaje* $\{a, b, c\}^*$*?*

Problema 18 *Simplificar la expresión regular* $\alpha = a+a(b+aa)(b^*(aa)^*)^*b^*+a(aa+b)^*$*, usando las equivalencias (semánticas) vistas en clase.*

Problema 19 *Calcular la derivada* $D_{ab}(\alpha) = D_a(D_b(\alpha))$*, siendo* $\alpha := a^*ab$.

Problema 20 *Comprobar que* $x := \alpha^*\beta$ *es una solución para la ecuación fundamental* $\alpha \equiv \alpha x + \beta$.

Problema 21 *Simplificar la expresión regular* $\alpha := 1^*01^* (01^*01^*0 + 1)^* 01^* + 1^*$.

Problema 22 *Hallar la expresión regular α asociada a la siguiente gramática (por el método de las ecuaciones lineales):*

$$S \to aA \mid cA \mid a \mid c,$$

$$A \to bS.$$

Aplicar el método de las derivadas a α y comparar los resultados.

Problema 23 *Hallar la gramática que genera el lenguaje descrito por la siguiente expresión regular:*

$$\alpha := (b + ab^{+}a)^{*}ab^{*}.$$

Problema 24 *Comprobar la equivalencia tautológica*

$$(b + ab^{*}a)^{*}ab^{*} \equiv b^{*}a(b + ab^{*}a)^{*}.$$

Problema 25 *Dada la expresión regular $\alpha := (ab+aba)^{*}$, hallar una gramática que genere el lenguaje $L(\alpha)$.*

Problema 26 *Dada la expresión regular $\alpha := a(bc)^{*}(b+bc)+a$, hallar una gramática G que genere el lenguaje $L(\alpha)$. Construir el sistema $S(G)$ asociado a la gramática calculada, resolverlo y comparar los resultados.*

Problema 27 *Hallar la expresión regular α asociada a la siguiente gramática:*

$$S \to bA \mid \lambda,$$

$$A \to bB \mid \lambda,$$

$$B \to aA.$$

Aplicar el método de las derivadas a α y comparar los resultados.

Problema 28 *Idem con la gramática:*

$$S \to 0A \mid 1B \mid \lambda,$$

$$A \to 1A \mid 0B,$$

$$B \to 1A \mid 0B \mid \lambda.$$

Problema 29 *Probar que si α es una expresión regular tal que $\alpha^2 \equiv \alpha$, entonces $\alpha^{*} \equiv \alpha + \lambda$.*

Problema 30 *Probar que si α es una expresión regular se cumple $\alpha^{*} \equiv \alpha^{*}\alpha + \lambda$.*

Problema 31 *Hallar dos expresiones regulares distintas que sean solución de la siguiente ecuación lineal*

$$(a + \lambda)X = X.$$

Problema 32 *Las Expresiones Regulares Avanzadas son expresiones regulares añadiendo diferentes operadores. Se utilizan en lenguajes de programación como Perl para búsquedas dentro de texto. Los operadores añadidos son los siguientes:*

- *operador de rango: Para las letras [a..z], significa que cualquier letra del rango es correcta.*
- *operador ?: este operador aplicado a una expresión regular entre paréntesis indica que necesariamente encaja en este esquema.*
- *operador /i: este operador, indica el símbolo en la posición i de la palabra, por ejemplo /1 indica el primer símbolo de la palabra.*

Mostrar como transformar las expresiones regulares avanzadas a expresiones regulares. Aplicarlo al siguiente caso,

$$[a..c]([C..E]^*)?/1.$$

Capítulo 3

Autómatas Finitos

Índice

3.1. Introducción: correctores léxicos o morfológicos

> ¿Ha probado con sus hijos a sentarse y darles una paliza?
>
> Bender B. Rodriguez

La siguiente etapa de nuestro recorrido son los autómatas finitos. Los autómatas finitos corresponden a correctores ortográficos. Se trata de la vieja tarea del maestro de primaria, corrigiendo los dictados, esto es, evaluando la presencia de errores ortográficos. El maestro no se ocupa de la corrección sintáctica del dictado (es él quien ha dictado las palabras y su secuencia, incluyendo signos ortográficos) sino solamente de los errores de transcripción y, por tanto, errores en la escritura morfológica o léxica. El otro ejemplo que mencionaremos viene del ADN, en este caso cada tres bases representa un aminoacido. Esto es lo que se llama el significado químico, ver [Martin-Mitrana-Paun, 04]. Conocer si una cadena de ADN tiene significado químico es de vital importancia para una celula y, por supuesto, esto debe de ser hecho de una forma eficiente.
En aplicaciónes informáticas, los autómatas finitos se usan para corregir (o señalar) los lugares en los que la morfología de un lenguaje (de programación) no ha sido respetada. Si, por ejemplo, alguien elabora un pequeño programa en C, Maple,

Matlab o, simplemente, un documento Textures, como parte del proceso de compilación existe un autómata finito que detecta la presencia de errores y, si encuentra alguno, salta mostrando dónde aparece. Otra aplicación de los autómatas finitos son los cortafuegos o firewalls. Para definir que ordenadores pueden usar que servicios, que comunicaciones se permiten dentro de una red... en general, se podría decir que aquellos componentes que tienen que ser eficientes y lo más sencillos posibles.
El gran impulsor de la teoría de autómatas fue J. von Neumann. Este matemático gastó buena parte de los últimos años de su vida en el desarrollo de la teoría de autómatas y, durante la Segunda Guerra Mundial, en el desarrollo de los computadores electrónicos de gran tamaño que fructificó en la aparición del ENIAC (un ordenador para calcular rápidamente trayectorias balísticas que fue financiado por el ejército de los Estados Unidos y finalizado en 1948)[1].

3.2. La noción de autómata

> Algo aquí tiene que estar mal, porque lo entiendo
>
> Alumno anónimo en una reunión

Veamos la primera aproximación de dispositivos que realizan computaciones. Formalmente, se definen como sigue:

Definición 42 *Llamaremos autómata finito indeterminístico a todo quíntuplo* $A := (Q, \Sigma, \delta, q_0, F)$ *donde:*

- Q *es un conjunto finito cuyos elementos se llaman estados y que suele denominarse espacio de estados,*
- Σ *es un conjunto finito (alfabeto),*
- q_0 *es un elemento de* Q *que se denomina estado inicial,*
- F *es un subconjunto de* Q*, cuyos elementos se denominan estados finales aceptadores,*
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \mapsto Q$ *es una correspondencia que se denomina función de transición.*

Si δ *es aplicación, el autómata se denomina determinístico.*

Nota 43 *En el caso indeterminístico, uno podría considerar la transición* δ *no como una correspondencia*

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \mapsto Q$$

sino como una aplicación

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \mapsto \mathcal{P}(Q),$$

[1] La historia del diseño y puesta en marcha del ENIAC y las personas involucradas puede seguirse en la página web *http://ftp.arl.mil/ ~mike/comphist/eniac-story.html* .

donde $\mathcal{P}(Q)$ es el conjunto de todos los subconjuntos del espacio de estados. Así, por ejemplo, si (q, a) no está en correspondencia (vía δ) con ningún estado, podríamos haber escrito $\delta(q, a) = \emptyset$.
Sin embargo, mantendremos la notación (incorrecta, pero cómoda) $\delta(q, a) = q'$ para indicar que el estado $q' \in Q$ están en correspondencia con el par (q, a) a través de la correspondencia δ. Así, por ejemplo, escribiremos $\neg\exists q' \in Q,\ \delta(q, a) = q'$ (o, simplemente, $\neg\exists\delta(q, a)$) para denotar que no hay ningún estado de Q en correspondencia con (q, a) a través de δ. Del mismo modo, $\exists q' \in Q,\ \delta(q, a) = q'$ (o, simplemente, $\exists\delta(q, a)$) en el caso contrario.

Como utilizaremos los autómatas finitos para hacer cálculos con palabras, es conveniente extender la definición de δ.

Definición 44 *Dado un autómata A en las notaciones anteriores y una palabra $x \in \Sigma^*$, definimos la correspondencia*

$$\delta^* : Q \times \Sigma^* \mapsto \mathcal{P}(Q)$$

dada por:

- $\delta^*(q, a) = \delta(q, a)$, *donde* $a \in \Sigma \cup \{\lambda\}$,
- $\delta^*(q, ax) = \{\delta^*(q', x) \mid \forall q' \in \delta(q, a)\ \}$.

Por abuso de notación, utilizaremos δ para denotar la aplicación δ^*, ya que las dos definiciones coinciden para letras.
Para ver la acción dinámica asociada a un autómata, como opera un autómata sobre palabras.

Definición 45 *Dado el autómata $A := (Q, \Sigma, q_0, F, \delta)$,*

- *$Q \times \Sigma^*$ es el conjunto de descripciones instantaneas,*
- *La transición $\Rightarrow$ se define por las reglas siguientes:*

$$(q, x) \Rightarrow (q', y) \ \Leftrightarrow\ \exists a \in \Sigma \cup \{\lambda\},\ \ x = ay,\ \ q' = \delta(q, a).$$

- *Denotaremos $(q, x) \Rightarrow (q', y)$ si exite una cadena de estados tales que,*

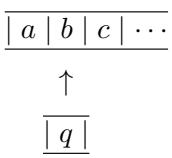
$$(q, x) \Rightarrow (q_1, w) \Rightarrow \ldots \Rightarrow (q', y).$$

- *Una descripción instantanea (q, λ) se dice final aceptadora si $q \in F$.*

Para interpretar mejor el proceso, hagamos nuestra primera descripción gráfica. Las palabras del alfabeto Σ^* se pueden imaginar como escritas en una cinta infinita, dividida en celdas en cada una de las cuales puedo escribir un símbolo de Σ.

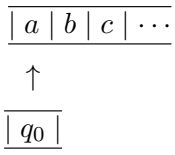
$$\underline{\overline{\mid a \mid b \mid c \mid \cdots}}$$

Hay una unidad de control que ocupa diferentes posiciones sobre la cinta y que dispone de una cantidad finita de memoria en la que puede recoger un estado de Q. Esta unidad de control solamente se moverá en una direccion, de izquierda a derecha.



Las descripciones instantaneas representan los contenidos de memoria (snapshot) del cálculo en curso, además la información de esta descripción permite continuar con el proceso. Así, dada una palabra $x = abc\cdots$ el autómata A computa sobre esta palabra de la manera siguiente:

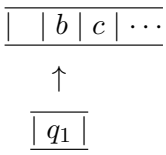
- *Inicializa (q_0, x), es decir*



- $q_1 \in \delta(q_0, a)$, *$y := bc\cdots$,*

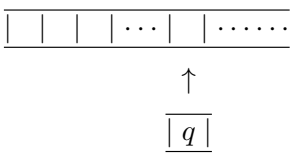
$$(q_0, x) \Rightarrow (q_1, y).$$

Gráficamente, borramos el contenido de la primera celda, cambiamos el estado en la unidad de control de q_0 (estado inicial) a q_1 y movemos la unidad de control un paso a la derecha:



- *El proceso continúa hasta que nos quedamos sin palabra, i.e. llegamos a la configuración (q, λ). Al final de este proceso, tenemos la sucesión de computación:*

$$(q_0, x) \Rightarrow (q_1, y) \to_A \cdots \Rightarrow (q, \lambda)$$



Notar que nuestras decisiones sabremos como, dado una palabra de entrada, podemos realizar una computación sobre ella.

Veamos ahora ejemplos de computación con un autómata finito y diferentes entradas.

Ejemplo 7 *Consideremos el siguiente autómata* $A := (Q, \Sigma, \delta, q_0, F)$ *donde,*

- $\Sigma := \{a, b\}$.
- $Q := \{q_0, q_1, q_2, q_3\}$.
- $F := \{q_2\}$.

Para la función de transición δ *elegiremos una representación a través de una tabla:*

δ	a	b
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_3	q_3

Esta tabla debe interpretarse como $\delta(q, a)$ *es el estado que aparece en la fila* q *y columna* a*. Revisemos la computación del autómata* A *sobre un par de entradas: Sea* $x = aabbb \in \Sigma^*$ *y veamos cómo funciona nuestro autómata:*

$$(q_0, aabbb) \Rightarrow (q_1, abbb) \Rightarrow (q_1, bbb) \Rightarrow (q_2, bb) \Rightarrow (q_2, b) \Rightarrow (q_2, \lambda),$$

por lo tanto $\delta^*(q_0, aabbb) = q_2$*. Mencionamos que la computación sobre la palabra* $x = aabbb$ *acaba en el estado* $q_2 \in F$*.*
Tomemos la palabra $y = baba \in \Sigma^*$ *y realicemos la computación* y*:*

$$(q_0, baba) \Rightarrow (q_3, aba) \Rightarrow (q_3, ba) \Rightarrow (q_3, a) \Rightarrow (q_3, \lambda),$$

por lo tanto $\delta^*(q_0, baba) = q_3$*. En este caso, la computación acaba con un estado* q_3 *que no es final.*

Representación gráfica de la función de transición.

> Enseñar ciencias de computación exige no infantilizar la materia con dibujitos
>
> E. Djisktra

Una forma estética, pero no siempre conveniente a la hora de manipular autómatas relativamente grandes, es la representación de autómatas finitos mediante grafos con aristas etiquetadas (pesos), un ejemplo se puede ver en la Figura 3.1. Las reglas son las siguientes:

- Los nodos del grafo están dados por los estados del grafo. Cada nodo está rodeado de, al menos, una circunferencia.

- Los nodos finales aceptadores del grafo son aquellos que están rodeados por dos circunferencias, el resto de los nodos aparecen rodeados de una sola circunferencia.
- Dada una transición $\delta(q, a) = q'$, asignaremos la arista del grafo (q, q') con etiqueta a.
- Hay una arista sin entrada, cuya salida es el nodo asociado al estado inicial.

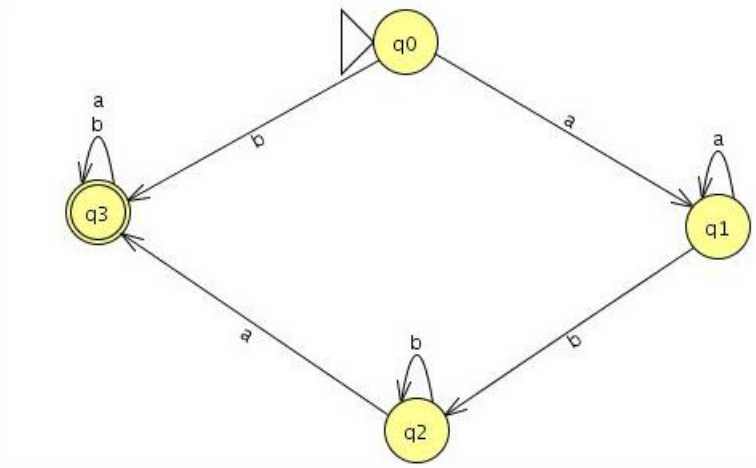


Figura 3.1: Representación gráfica del autómata.

Usaremos más habitualmente la representación de las funciones de transición bien mediante listas o bien mediante tablas. Como hicimos para las expresiones regulares, relacionaremos un lenguaje con cada autómata.

Definición 46 *Llamaremos lenguaje aceptado por un autómata A al conjunto de palabras $x \in \Sigma^*$ tales que $\delta^*(q_0, x) \in F$.*

En términos de la Definición 44, podremos también escribir:

$$L(A) := \{x \in \Sigma^* \mid \delta^*(q, x) \cap F \neq \emptyset\}.$$

Podemos interpretar un autómata como un evaluador de la función característica de un subconjunto de $L \subseteq \Sigma^*$:

$$\chi_L : \Sigma^* \mapsto \{0, 1\}.$$

El algoritmo 3 muestra como evaluar si una palabra está en el lenguaje aceptado por un autómata finito determinista. Para los autómatas indeterministas hay que trabajar un poco más para evaluar χ_L, aunque siempre es posible encontrar un autómata equivalente determinista.
Una buena referencia sobre autómatas es el texto [Davis-Weyuker, 94], donde también se pueden encontrar ejemplos sencillos que ayuden al alumno a asimilar la noción.

Algorithm 3 Algoritmo para evaluar la función caracteristica de un lenguaje aceptado por un autómata finito determinista

Input: A autómata determínista y una palabra x
Output: 1 si $x \in L(A)$, 0 en caso contrario

```
q := δ*(q0, x)
if q ∈ F then
  return 1
else
  return 0
end if
```

3.3. Determinismo e Indeterminismo

> Te voy a cantar la canción más bella que conozco. A quien escucha esta canción no puede resistir llorar o.... si que puede resistir llorar
>
> "Alicia a traves del espejo"

Una manera bastante natural de interpretar el autómata finito es usar un pseudo–código para expresar un autómata como un programa con un **while**. Informalmente, sea $A := (Q, \Sigma, \delta, q_0, F)$ un autómata. El programa (algoritmo) que define es el dado por la siguiente descripción:

Algorithm 4 El autómata como programa

Input: $x \in \Sigma^*$

```
I := (q0, x)
while I ∉ F × {λ} do
  if I = (q, ay), where a ∈ Σ ∪ {λ} and ay ≠ λ then
    I := (δ(q0, a), y)
  else
    return 0
  end if
end while
return 1
```

Nótese que hemos introducido deliberadamente un pseudo–código que no necesariamente termina en todos los inputs. Esto es por analogía con las máquinas de Turing y el estudio de los lenguajes recursivamente enumerables y recursivos. Aquí, el pseudo–código tiene una interpretación directa y natural en el caso determinístico y genera una forma imprecisa en el caso indeterminístico. Esta interpretación como programa (determinístico) de este pseudo–código depende esencialmente de la ausencia de dos obstrucciones:

- La presencia de $\lambda-$transiciones, esto es, de transiciones de la forma $\delta(q, \lambda)$ que

pueden hacer que caigamos en un ciclo infinito.

- La indefinición de $I = \delta(\delta(q, a), y)$ por no estar definido $\delta(q, a)$ o por tener más de un valor asociado.

Ambas obstrucciones se resuelven con los algoritmos que se describen a continuación. Definamos de una manera precisa que queremos decir con $\lambda-$transiciones.

Definición 47 *Se denominan $\lambda-$transiciones a las transiciones de una autómata $A := (Q, \Sigma, \delta, q_0, F)$ de la forma:*

$$\delta(q, \lambda) = q'.$$

Un autómata se dice libre de $\lambda-$transiciones si no tiene ninguna $\lambda-$transición.

En un sentido menos preciso, las $\lambda-$transiciones son meras transformaciones de los estados conforme a reglas que no dependen del contenido de la cinta.
Para cada descripción instantanea (q, x) en el sistema de transición asociado al autómata y supuesto que existe una $\lambda-$transición $\delta(q, \lambda) = q'$, entonces la transición será de la forma $(q, x) \Rightarrow (q', x)$, donde x no es modificado y sólo hemos modificado el estado.
En términos de operaciones de lecto–escritura, nuestra $\lambda-$transición realiza las siguientes tareas[2]:

- *NO lee* el contenido de la cinta.
- *Modifica* el estado en la unidad de control.
- *NO borra* el contenido de la celda señalada por la unidad de control.
- *NO se mueve* a la derecha.

Notese que las $\lambda-$transiciones se pueden interpretar un "indeterminismo" en el estado actual. Lo que implica que a partir de las $\lambda-$transiciones de un autómata podemos construir un grafo representando los estados entre los que nos podemos mover por medio de $\lambda-$transiciones. Dado un autómata $A := (Q, \Sigma, \delta, q_0, F)$, definimos el grafo de las $\lambda-$transiciones de A mediante $G := (V, E)$, donde las reglas son:

- $V := Q$.
- Dados $q, q' \in V$, decimos que $(q, q') \in E$ si $q' \in \delta(q, \lambda)$, i.e.

$$E := \{(q, q') \mid q' \in \delta(q, \lambda)\}.$$

Si miramos el grafo asociado al autómata 3.2, podemos extraer el grafo de $\lambda-$transiciones, dejando los mismos nodos (o vértices) y suprimiendo todas las aristas que estén etiquetadas con algún símbolo del alfabeto (y dejando solamente las que están etiquetadas con λ).

[2] o no realiza, depende de donde quiera poner el acento el lector.

A partir del grafo de las $\lambda-$transiciones podemos considerar la $\lambda-$*clausura* de un nodo (estado), definiéndola del modo siguiente:

$$\lambda - cl(q) := \{q' \in V \mid (q, \lambda) \Rightarrow^* (q', \lambda)\}.$$

Obsérvese que la $\lambda-$clausura de un nodo q está determinada por las descripciones instantaneas (con palabra vacía λ) alcanzables desde la descripción autómatica (q, λ). Obsérvese también que la palabra vacía λ está en el lenguaje aceptado $L(A)$ si y solamente si la clausura $\lambda - cl(q_0)$ del estado inicial contiene algún estado final (i.e. $\lambda - cl(q_0) \cap F \neq \emptyset$).
Del mismo modo, dados $q \in Q$ y $a \in \Sigma$, definiremos la $\lambda-$clausura de q y a mediante:

$$\lambda - cl(q, a) := \{q' \in V \mid (q, \lambda) \Rightarrow^* (q', \lambda),\ \exists q'' \in \lambda - cl(q),\ q' \in \lambda - cl(\delta(q'', a))\}.$$

Nuestro objetivo es probar el siguiente enunciado:

Proposición 48 *Para cualquier lenguaje L que sea aceptado por un autómata con $\lambda-$transiciones, entonces existe un autómata libre de $\lambda-$transiciones que acepta el mismo lenguaje. Más aún, la transformación de un autómata a otra se puede realizar algorítmicamente.*

Demostración. Nos basta con tomar como dado de entrada un autómata $A := (Q, \Sigma, \delta, q_0, F)$ y definir un nuevo autómata que elimina las $\lambda-$transiciones. El nuevo autómata no ha de ser determinista, pero éso es irrelevante como veremos en la Proposición 51.
Construiremos un nuevo autómata $B := (Q', \Sigma, \delta', q_0', F')$ definido conforme al Algoritmo 3.3.

Algorithm 5 Algoritmo para hallar un autómata equivalente sin λ-transiciones

Input: autómata $A := (Q, \Sigma, \delta, q_0, F)$
Output: retorna un automata $B = (Q', \Sigma, \delta', q_0, F')$

```
Q' := Q
for q ∈ Q do
    Calcular λ − cl(q)
end for
F' := F ∪ {q | λ − cl(q) ∩ F ≠ ∅}
for q ∈ Q do
    if λ − cl(q, a) ≠ ∅ then
        δ'(q, a) = λ − cl(q, a)
    end if
end for
return B := (Q', Σ, δ', q0', F')
```

Nótese que $\delta'(q, \lambda)$ no está definida para ningún $q \in Q$. Dejamos como ejercicio la comprobación de que el autómata B acepta L. ∎

Nota 49 *Obsérvese que el resultado de eliminar $\lambda-$transiciones no tiene porque dar un autómata determinista. Curiosamente, parece que el autómata resultante puede tener una expresión más complicada que el autómata inicial.*

Nota 50 *Nótese que en el caso en que $\lambda \in L(A)$ (i.e. $\lambda - cl(q_0) \cap F \neq \emptyset$), el estado inicial pasa a ser también estado final aceptador.*

Determinismo e Indeterminismo en Autómatas

> Todo está determinado, Dios no juega a los dados
>
> Albert Einstein

Una primera preocupación técnica podría ser el papel que juega el indeterminismo en la clase de lenguajes aceptados por autómatas. Los siguientes resultados tranquilizan mostrando que el indeterminismo es irrelevante en cuanto a la clase de lenguajes aceptados.

Proposición 51 *Si un lenguaje $L \subseteq \Sigma^*$ es aceptado por un autómata finito indeterminista, entonces, existe un autómata finito determinista que lo acepta.* [3]

Demostración. La idea es simple, sea $A := (Q, \Sigma, \delta, q_0, F)$ un autómata indeterminista sin $\lambda-$transiciones que acepta un lenguaje $L \subseteq \Sigma^*$. Definamos el siguiente autómata determinista B dado por:

- $Q' := \mathcal{P}(Q)$ (el espacio de estados es el conjunto de las partes de Q).
- $F' := \{X \in Q' \mid X \cap F \neq \emptyset\}$ (las configuraciones finales aceptadoras son aquellas que contienen algún estado del espacio F de estados finales aceptadores).
- $q_0' := \{q_0\}$ (el conjunto formado por la antigua configuración inicial).
- La función de transición
$$\delta' : Q' \times \Sigma \mapsto Q'$$
definida mediante:
$$\delta'(X, a) := \{q' \in Q \mid \exists q \in X,\ q' \in \delta(q, a)\}.$$

Dejamos el asunto de la comprobación como ejercicio. ∎

Nota 52 *A partir de ahora usaremos autómatas deterministas e indeterministas sin la preocupación sobre el indeterminismo, dado que podemos reemplazar unos por otros sin mayores problemas.*

[3] Una característica del indeterminismo es que no modifica la clase de lenguajes aceptados; aunque sí podría modificar los tiempos de cálculo. Esto no afecta a los autómatas finitos, según se prueba en este enunciado, pero sí está detrás de la Conjetura de Cook **P** = **NP**?.

3.4. Lenguajes regulares y autómatas

> Las cosas cambian para seguir igual
>
> Jose Ortega y Gasset

Como indica el título y la cita, el objetivo de esta sección es mostrar que los lenguajes aceptados por los autómatas son los lenguajes regulares y si bien, hemos tenido muchas definiciones, al final llegaremos a que todos los conceptos son equivalentes. Para ello, mostraremos dos procedimientos de paso conocidos como Teorema de Análisis y Teorema de Síntesis de Kleene (cf. [Kleene, 56]).

Teorema de Análisis de Kleene

Nuestra primera duda es si cualquier lenguaje aceptado por un autómata finito está generado por una expresión regular. El siguiente teorema afirma eso y además da un algoritmo para calcular una expresión regular que genera el mismo lenguaje. Se deja al alumno el ejercicio de demostrar la complejidad del algoritmo.

Teorema 53 *Sea $L \subseteq \Sigma^*$ un lenguaje aceptado por un autómata finito determinista. Entonces, existe una expresión regular α sobre el alfabeto Σ tal que $L = L(\alpha)$. Más aún, mostraremos que existe un procedimiento tratable que permite calcular la expresión regular asociada al lenguaje aceptado por un autómata.*

Demostración. Nos limitaremos con mostrar el procedimiento, que casi viene prefigurado por las definiciones.
Para ello construiremos un sistema de ecuaciones lineales en expresiones regulares con las reglas siguientes:

- Supongamos que $Q := \{q_0, \ldots, q_n\}$. Introducimos un conjunto de variables biyectable con Q dado por $\{Y_0, Y_1, \ldots, Y_n\}$. La biyección será dada por $q_0 \mapsto Y_0, \ldots, q_n \mapsto Y_n$.
- Definimos un sistema de ecuaciones lineales en expresiones regulares:

$$\begin{pmatrix} Y_0 \\ Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} & \cdots & \alpha_{0,n} \\ \vdots & \ddots & \vdots \\ \alpha_{n,0} & \cdots & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} Y_0 \\ Y_1 \\ \vdots \\ Y_n \end{pmatrix} + \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{pmatrix},$$

  Conforme a las reglas siguientes:

  - Para cada i, $0 \leq i \leq n$, definamos $\beta_i := \lambda$ si $q_i \in F$ y $\beta_i := \emptyset$ en caso contrario.
  - Para cada i, j, $0 \leq i, j \leq n$, definamos $\alpha_{i,j}$ mediante:

$$A_{i,j} := \{a \in \Sigma \mid \delta(q_i, a) = q_j\}.$$

Definiremos

$$\alpha_{i,j} := \sum_{a \in A_{i,j}} a,$$

notando que si $A_{i,j} = \emptyset$, entonces, $\alpha_{i,j} := \emptyset$.

Entonces, si $(\omega_0, \ldots, \omega_n)$ es una solución del anterior sistema lineal, entonces el lenguaje que define ω_0 es el lenguaje aceptado por el autómata. La idea de la demostración es la siguiente: empecemos por calcular el lenguaje de las palabras que empezando en q_0 son aceptadas por el autómata y llamemos a la expresión regular que genera este lenguaje ω_0. De la misma forma, para cada uno de los estados asociamos una expresion regular $\omega_1, \ldots, \omega_n$. Hay una clara relación entre estos lenguajes, que está dada por las ecuaciones lineales dadas más arriba. El lenguaje que describe ω_0 está claramente formado por la unión de los lenguajes describidos por las expresiones regulares ω_i correspondientes, con prefijo dado por el símbolo de la transición. Además, si el estado es final hay que añadir la palabra λ. ∎

Definición 54 (Sistema Característico de un Autómata) *Se denomina sistema de ecuaciones característico de un autómata al sistema de ecuaciones lineales en expresiones regulares obtenido conforme a las reglas descritas en la demostración del Teorema de Análisis de Kleene.*

Nota 55 *Nótese que, a partir del Sistema característico de un autómata A uno podría reconstruir una gramática regular G que genera el mismo lenguaje $L(G)$ que el aceptado por A, i.e. $L(G) = L(A)$.*

Teorema de Síntesis de Kleene

> Es más difícil que volver a meter los gusanos en una lata
>
> Proverbio ingles

Isaac Asimov escribio una vez un cuento en el que los hombres le preguntaban a un supercomputador como revertir la disminución de la entropia y devolver el universo a un estado primigenio. Normalmente realizar una tarea que implique transformar una cosa en otra no implica inmediatamente poder realizar la tarea inversa con la misma facilidad. Pero, curiosamente, para los lenguajes regulares es fácil encontrar diferentes representaciones del mismo lenguaje.
En esta segunda parte, vamos a mostrar el paso inverso del teorema de análisis de Kleene. Esto es, que para cualquier lenguaje descrito por una expresión regular se puede encontrar un autómata determinista que lo acepta. Para ello usaremos el árbol de formación de la expresión regular. Comenzaremos por un sencillo lema.

Lema 56 *Dado un lenguaje L aceptado por un autómata, existe un autómata $A := (Q, \Sigma, \delta, q_0, F)$ que acepta L y que verifica las siguientes propiedades:*

a. $\sharp(F) = 1$, *es decir, sólo hay una configuración final aceptadora. Supondremos* $F := \{f\}$.

b. $\delta(q, a)$ *está definida para todo* $q \in Q$ *y todo* $a \in \Sigma$.

c. *Las únicas* $\lambda-$*transiciones entran en* f. *Es decir,*

$$\text{Si } \delta(q', \lambda) = q \Leftrightarrow q = f.$$

Demostración. Dado el autómata $A := (Q, \Sigma, \delta, q_0, F)$, que podemos suponer determinista, definamos el nuevo autómata $B := (Q', \Sigma, \delta', q_0, F')$ conforme a las reglas siguientes:

- Sea $f, ERROR$ dos nuevos estados tal que $f, ERROR \notin Q$. Definamos $Q' := Q \cup \{f\} \cup \{ERROR\}$.
- Definamos $F' := \{f\}$.
- Para cada $q \in Q$ y para cada $a \in \Sigma$, definamos para los nuevos estados

$$\delta'(ERROR, a) := ERROR, \qquad \delta'(f, a) := ERROR$$

  y extendamos la función de transición para los antiguos estados si $a \in \Sigma$

$$\delta'(q, a) := \begin{cases} \delta(q, a), \text{ si } \delta(q, a) \text{ está definida,} \\ ERROR, \text{ en otro caso.} \end{cases}$$

- Para cada $q \in F$, definamos $\delta'(q, \lambda) := f$.

Está claro que B acepta el mismo lenguaje que aceptaba A. La razón es simple: la única manera de alcanzar el nuevo estado f es llegar a un estado final con la cinta vacía. ■

Teorema 57 *Sea* Σ *un alfabeto finito y* α *una expresión regular sobre* Σ. *Entonces, existe un autómata finito* A *que reconoce el lenguaje descrito por* α. *Más aún, el proceso de obtención del autómata a partir de la expresión regular se puede lograr de manera algorítmica.*

Demostración. De nuevo nos limitaremos a describir un proceso algorítmico que transforma expresiones regulares en autómatas, usando los operadores de definición de la expresión (i. e., el procedimiento es recursivo en la construcción de la expresión regular).

- **El caso de los símbolos primarios:**
  - *El caso* $\emptyset$*:* Bastará un autómata donde el conjunto de estados esta definido por $Q := \{q_0, f\}$, $F := \{f\}$ tal que la función de transición no esté definida en ningún caso.

- *El caso λ:* De nuevo usaremos $Q := \{q_0, f\}$, $F := \{f\}$, pero la función de transición está definida solamente para $\delta(q_0, \lambda) = f$ y no definida en el resto de los casos.
- *El caso constante $a \in \Sigma$:* Igual que en el caso anterior, usaremos $Q := \{q_0, f\}$, $F := \{f\}$, pero la función de transición está definida solamente para $\delta(q_0, \lambda) = f$ y no definida en el resto de los casos.

▪ **Siguiendo los operadores:**

- *El autómata de la unión ($\alpha + \beta$):* Si tenemos $A := (Q, \Sigma, \delta, q_0, F)$ un autómata determinista que acepta $L(\alpha) \subseteq \Sigma^*$ y un segundo autómata también determinista $B := (Q', \Sigma, \delta', q_0', F')$ que acepta $L(\beta) \subseteq \Sigma^*$, definimos un nuevo autómata[4] $C := (Q'', \Sigma, \delta'', q_0'', F'')$ que acepta $L(\alpha) \cup L(\beta)$ y viene dado por las reglas siguientes:
  - $Q'' := Q \times Q'$,
  - $F'' := (F \times Q') \cup (Q \times F')$
  - $q_0'' := (q_0, q_0')$
  - $\delta''((q, q'), a) = (\delta(q, a), \delta'(q', a))$, $\forall q \in Q, q' \in Q'$ y $\forall a \in \Sigma \cup \{\lambda\}$.
- *El autómata de la concatenación ($\alpha \cdot \beta$):* Supongamos $A := (Q, \Sigma, \delta, q_0, F)$ un autómata que acepta $L(\alpha) \subseteq \Sigma^*$ y un segundo autómata $B := (Q', \Sigma, \delta', q_0', F')$ que acepta $L(\beta) \subseteq \Sigma^*$. Supongamos que A verifica las condiciones descritas en el Lema 56 y sea $F' := \{f\}$.
  Definimos un nuevo autómata $C := (Q'', \Sigma, \delta'', q_0'', F'')$ que acepta el lenguaje descrito por la expresión regular $\alpha \cdot \beta$ y viene dado por las reglas siguientes:
  - $Q'' := (Q \times \{1\}) \cup (Q' \times \{2\})$.
  - $F'' := F' \times \{2\}$.
  - $q_0'' := (q_0, 1)$
  - La función de transición $\delta'' : Q'' \times (\Sigma \cup \{\lambda\}) \mapsto Q''$, viene dada por:

$$\delta''((q,i),a) := \begin{cases} (\delta(q,a),1)\,, & \text{si } q \in Q,\ i = 1 \\ (q_0',2)\,, & \text{si } q = f \in F,\ i = 1,\ a = \lambda \\ (\delta'(q,a),2)\,, & \text{si } q \in Q',\ i = 2 \end{cases} \tag{3.1}$$

- *El autómata del lenguaje descrito por ($L(\alpha)^*$):* De nuevo suponemos que tenemos un autómata $A := (Q, \Sigma, \delta, q_0, F)$ que acepta el lenguaje $L(\alpha)$. Podemos suponer que dicho autómata verifica las condiciones del Lema 56. Supongamos $F = \{f\}$. Definamos un nuevo autómata $B := (Q', \Sigma, \delta', q_0', F')$ añadiendo un nuevo estado inicial al autómata q_0' y definiendo $Q' := Q \cup q_0'$, $F' := \{q_0'\}$. La función de transición se define conforme a las reglas siguientes:

$$\delta'(q,a) := \begin{cases} \delta(q,a) & \text{si } q \in Q' \setminus \{q_0',\}, \\ q_0' & \text{si } q = f, a = \lambda, \\ q_0 & \text{si } q = q_0', a = \lambda, \end{cases}$$

[4] Esta construcción se la conoce como *Autómata Producto.*

Está claro que este autómata acepta el lenguaje previsto.

Con esto acabamos la demostración, ya que cualquier expresión regular esta formada por concatenación, suma de expresiones regulares o la operación estrella de una expresión regular. ■

3.5. Lenguajes que no son regulares

A conjecture both deep and profound
Is whether a circle is round.
In a paper of Erdős
Written in Kurdish
A counterexample is found

Leo Moser

Es el sentimiento de los autores, que este capítulo no fuera necesario. En verdad que la vida sería más sencilla si todos los lenguajes fueran regulares y todos los autómatas que se utilizarán fueran finitos. Pero esto no es así.
La tradición usa el Lema de Bombeo para mostrar las limitaciones de los lenguajes regulares. El resultado es debido a Y. Bar-Hillel, M. Perles, E. Shamir[5]. Este Lema se enuncia del modo siguiente:

Teorema 58 (Pumping Lemma) *Sea L un lenguaje regular. Entonces, existe un número entero positivo $n \in \mathbb{N}$ tal que para cada palabra $x \in L$, con $|x| \geq n$ existen $y, w, z \in \Sigma^*$ verificando los siguientes propiedades:*

- $|w| \geq 1$,
- $|yw| \leq n$,
- $x = ywz$,
- *Para todo $\ell \in \mathbb{N}$, las palabras $yw^\ell z \in L$.*

El Lema de Bombeo simplemente dice que hay prefijos y una lista finita de palabras tal que, bombeando esas palabras, generaremos palabras del mismo lenguaje regular.

Nota 59 *Hay varias razones por las que éste es un enunciado insuficiente. La primera es estética: un exceso de fórmulas cuantificadas hace desagradable su lectura. Adicionalmente, debemos señalar que el Lema de Bombeo da una condición necesaria de los lenguajes regulares, pero no es una condición suficiente. Es decir, hay ejemplos de lenguajes que no son regulares (ver Corolario 62) pero que satisfacen el Lema de Bombeo (ver ejemplo 8 más abajo)*

[5] Y. Bar-Hillel, M. Perles, E. Shamir. "On formal properties of simple phrase structure grammars". *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung* **14** (1961) 143–172.

Ejemplo 8 *Los lenguajes regulares satisfacen el Lema de Bombeo y también lo satisface el siguiente lenguaje:*

$$L := \{a^i b^j c^k \mid [i = 0] \vee [j = k]\} \subseteq \{a, b, c\}^*.$$

Veamos que satisface el Teorema 58 anterior con $p = 1$. Para ello, sea $x \in L$ y tendremos tres casos:

- *Caso 1: $i = 0$ con $j = 0$ o, lo que es lo mismo, $x = c^k$, con $k \geq 1$. En ese caso, tomando $y = \lambda$, $w = c$, $z = c^{k-1}$, tenemos que $yw^\ell z \in L$, $\forall \ell \in \mathbb{N}$.*
- *Caso 2: $i = 0$ con $j \geq 1$ o, lo que es lo mismo, $x = b^j c^k$, $j \geq 1$. En ese caso, tomando $y = \lambda$, $w = b$, $z = b^{j-1}c^k$, tenemos que $yw^\ell z \in L$, $\forall \ell \in \mathbb{N}$.*
- *Caso 3: $i \geq 1$ o, lo que es lo mismo, $x = a^i b^j c^j$. En ese caso, tomando $y = \lambda$, $w = a$, $z = a^{i-1}b^j c^j$, tenemos que $yw^\ell z \in L$, $\forall \ell \in \mathbb{N}$.*

Veremos más adelante (Corolario 62) que este lenguaje no es regular.

Definición 60 (Prefijos) *Sea Σ un alfabeto finito y sea $L \subseteq \Sigma^*$ un lenguaje cualquiera. Definimos la relación de prefijos la siguiente relación de equivalencia sobre Σ^*: dados $x, y \in \Sigma^*$, $x \sim_L y$ si y solamente si:*

$$\forall w \in \Sigma^*, xw \in L \Leftrightarrow yw \in L.$$

Verificar que estamos ante una relación de equivalencia es un mero ejercicio. Lo que pretendemos es caracterizar los lenguajes aceptados por un autómata mediante una caracterización del conjunto cociente: $\Sigma^*/\sim_L$.

Teorema 61 (Myhill–Nerode) [6] *Si $L \subseteq \Sigma^*$ es un lenguaje, entonces L es regular si y solamente si $\Sigma^*/\sim_L$ es finito.*

Demostración. Comencemos con una de las implicaciones. Supongamos que L es el lenguaje aceptado por un autómata determinista $A := (Q, \Sigma, \delta, q_0, F)$. Consideremos el conjunto de los estados alcanzables por alguna computación de A:

$$Q' := \{q \in Q \mid \exists x \in \Sigma^*,\ (q_0, x) \Rightarrow^* (q, \lambda)\}$$

está claro que $Q' \subseteq Q$ es un conjunto finito. Para cada $q \in Q'$, sea $x_q \in \Sigma^*$ una palabra cualquiera tal que $(q_0, x_q) \Rightarrow^* (q, \lambda)$. Sea el siguiente conjunto

$$\mathcal{A} := \{x_q \mid q \in Q'\}$$

Claramente $\mathcal{A}$ es un conjunto finito y vamos a probar que

$$\Sigma^*/\sim_L = \{[x_q] \mid x_q \in \mathcal{A}\},$$

donde $[x_q]$ es la clase de equivalencia definida por x_q y tendremos la afirmación. Ahora, tomemos $x \in \Sigma^*$ y sea $(q_0, x) \Rightarrow^* (q, \lambda)$, $q \in Q'$. Para cualquier $y \in \Sigma^*$,

[6] Rabin, M. and Scott, D.."Finite automata and their decision problems". *IBM Journal of Research & Development* **3** (1959), 114-125.

el sistema de transición asociado al autómata A, trabajando sobre xy realiza la siguiente computación:

$$(q_0, xy) \Rightarrow \cdots \Rightarrow (q, y)$$

mientras vamos borrando la x. Ahora bien, si tomamos $x_q y \in \Sigma^*$, el cálculo hará también el camino:

$$(q_0, x_q y) \Rightarrow \cdots \Rightarrow (q, y)$$

Lo que pase a partir de (q, y) es independiente de por dónde hayamos empezado, luego xy es aceptado por A si y solamente si $x_q y$ es aceptado por A. Con esto hemos demostrado una de las direcciones del enunciado, esto es, si el lenguaje es regular y, por ende, aceptado por un autómata finito, entonces, el conjunto cociente $\Sigma^*/\sim_L$ es finito.
Para el recíproco, supongamos que $\Sigma^*/\sim_L$ es finito y supongamos:

$$\Sigma^*/\sim_L = \{[x_0], \ldots, [x_n]\},$$

donde $x_0, \ldots, x_n \in \Sigma^*$. Podemos suponer que $x_0 = \lambda$ (la palabra vacía estará en alguna clase de equivalencia). Ahora definamos un autómata $A := (Q, \Sigma, \delta, q_0, F)$ con las reglas siguientes:

- Los estados están definidos mediante:

$$Q := \{q_0, \ldots, q_n\} \iff q_i = [x_i].$$

- El estado inicial es dado por $q_0 = [x_0] = [\lambda]$.
- El espacio de estados finales aceptadores es $F := \{q_i \mid x_i \in L\}$.
- La función de transición es dada para cada $a \in \Sigma \cup \{\lambda\}$, mediante:

$$\delta(q_i, a) := q_j, \quad \text{donde } q_j := [x_i a].$$

La definición de la función de transición muestra claramente que el autómata es determinista, veamos que realiza la tarea indicada. La configuracion inicial es (q_0, y) para cualquier palabra $y \in \Sigma^*$ y no es difícil probar que el resultado de la computación sobre este autómata finito es q_i donde $[x_i] = [y]$. Por la definición de la relación, sabemos que $y \in L \iff x_i \in L$, y esto acaba la demostración. ∎

Veamos ahora una aplicación inmediata de este resultado que hemos demostrado.

Corolario 62 *El lenguaje L descrito en el Ejemplo 8 no es un lenguaje regular.*

Demostración. Basta con verificar que no satisface las propiedades descritas en el Teorema de Myhill–Nerode. Para ello supongamos que el conjunto cociente $\Sigma^*/\sim_L$ es finito, es decir,

$$\Sigma^*/\sim_L := \{[x_0], \ldots, [x_n]\}.$$

Consideremos la sucesión infinita $y_i := ab^i$. Como sólo hay un número finito de clases de equivalencia, hay una clase de equivalencia (digamos $[x_0]$) que contiene

una infinidad de términos de esa sucesión. En otras palabras, existe una sucesión infinita y creciente:

$$1 < n_1 < n_2 < \cdots < n_i < n_{i+1} < \cdots , \tag{3.2}$$

de tal modo que

$$\{y_{n_i} \mid i \in \mathbb{N}\} \subseteq [x_0].$$

En este caso, se ha de tener, además, la siguiente propiedad:

$$\forall w \in \Sigma^*,\ x_0 w \notin L. \tag{3.3}$$

Para probarlo, nótese que si existiera alguna palabra $w \in \Sigma^*$ tal que $x_0 w \in L$, w es de longitud finita. Supongamos $m := |w| \in \mathbb{N}$ esa longitud. Como la sucesión de los n_i's es infinita, ha de existir algún n_t tal que $n_t > m + 1$.
Pero, además, $ab^{n_t} \sim_L x_0$, luego, como $x_0 w \in L$, entonces también se ha de tener

$$ab^{n_t} w \in L.$$

Por la definición de L tendremos, entonces que

$$ab^{n_t} w = ab^j c^j,$$

para algún j. Obviamente ésto significa que w es de la forma $w = b^k c^j$ y, necesariamente, $j = n_t + k \geq n_t$. Por lo tanto, $m + 3 = |w| + 3 \geq j + 3 = n_t + k + 3 > n_t$, contraviniendo nuestra elección de $n_t \geq m + 3$.
Con esto hemos probado la veracidad de la afirmación (3.3) anterior. Pero, de otro lado, $ab^{n_1} c^{n_1} \in L$ y $ab^{n_1} \sim_L x_0$ luego $x_0 c^{n_1} \in L$, lo que contradice justamente la afirmación probada. La hipótesis que no se sostiene es que el conjunto cociente $\Sigma^* / \sim_L$ sea finito y, por tanto, L no es un lenguaje regular. ∎

El Palíndromo no es un lenguaje regular

> Si he logrado ver más lejos, ha sido porque he subido a hombros de gigantes
>
> Isaac Newton

Se trata del ejemplo clásico y común que "deben" contemplar todos los cursos de Introducción a los lenguajes regulares: *el Palíndromo* o, en buen catalán, el problema de la detección de los *"cap–i–cua"*, del que veremos que no es un lenguaje regular, como consecuencia del resultado de Myhill y Nerode anterior.
Comencemos recordando la definición del Palíndromo ya presentado en Secciones anteriores.

- El reverso de la palabra vacia es la palabra vacía $\lambda^R := \lambda$,

- en cambio, si la palabra no es vacía entonces $x := ay \in \Sigma^*$, donde $a \in \Sigma$ y $y \in \Sigma^*$ y se tiene
$$x^R := y^R a.$$

El lenguaje del Palíndromo es dado por las palabras que coinciden con su reverso, esto es,
$$\mathcal{P} := \{x \in \Sigma^* \mid x^R = x\}.$$

Daremos una demostración del resultado siguiente usando la finitud de los prefijos.

Corolario 63 *El Palíndromo no es un lenguaje regular si el alfabeto tiene al menos dos letras distintas.*

Demostración. Por simplicidad supongamos $\Sigma = \{a, b\}$. Para cada número natural $i \in \mathbb{N}$, consideremos la palabra de longitud $i + 2$ siguiente:
$$y_i := a^i ba.$$

Supongamos que el palíndromo es un lenguaje regular y será finito el conjunto cociente siguiente:
$$\Sigma^*/\sim_{\mathcal{P}} = \{[x_0], \ldots, [x_n]\}.$$

Por otro lado, consideremos las clases definidas por los elementos de la sucesión anterior:
$$\mathcal{A} := \{[y_i] \mid i \in \mathbb{N}\}.$$

Como el conjunto cociente es finito, el anterior conjunto $\mathcal{A}$ es finito y, por tanto, habrá alguna clase (supongamos otra vez que corresponde a $[x_0]$) en la que estará una infinitud de elementos de la sucesión $\{y_i \mid i \in \mathbb{N}\}$. Es decir, que existe una sucesión infinita creciente de índices:
$$n_1 < n_2 < n_3 < \cdots < n_i < \cdots$$

de tal modo que $y_{n_i} \in [x_0]$. Supongamos n_t suficientemente grande (por ejemplo, $n_t \geq 2|x_0| + 3$).
Ahora obsérvese que $y_{n_t} y_{n_t}^R \in \mathcal{P}$ es un palíndromo. Como $y_{n_t} \sim_{\mathcal{P}} x_0$ (están en la misma clase de equivalencia), tendremos que $x_0 y_{n_t}^R \in \mathcal{P}$. Por tanto,
$$x_0 y_{n_t}^R = y_{n_t} x_0^R. \tag{3.4}$$

Como la longitud de y_{n_t} es mayor que la de x_0, tendremos que x_0 debe coincidir con los primeros $m = |x_0|$ dígitos de y_{n_t}. Por tanto, $x_0 = a^m$.
Ahora bien, el único símbolo b de la palabra $x_0 y_{n_j}^R$ en la identidad (3.4) ocupa el lugar $m+2$, mientras que el único símbolo b de la palabra $y_{n_j} x_0^R$ ocupa el lugar n_j+1, como $n_j \geq 2m + 3$ no es posible que ambas palabra sean iguales, contradiciendo la igualdad (3.4) y llegando a contradicción. Por tanto, el palídromo no puede ser un lenguaje regular. ∎

En realidad, el conjunto de los lenguajes regulares es pequeño. La mayoria de los lenguajes complicados están en niveles superiores de la jerarquía de Chomsky. En el siguiente ejemplo vemos una pequeña muestra de ellos.

Ejemplo 9 *Los siguientes son también ejemplos de lenguajes no regulares:*

- $\Sigma = \{0, 1\}$ *y el lenguaje* L *dado por la condición el número de* 1*'s es mayor que el número de* 0*'s.*
- *Para el mismo alfabeto el lenguaje:*

$$L := \{0^m 1^m \mid m \in \mathbb{N}\}$$

- *Para el alfabeto* $\Sigma = \{0, 1, \ldots, 9\}$ *sea* $\pi \subseteq \Sigma^*$ *el lenguaje formado por las palabras que son prefijos de la expansión decimal de* $\pi \in \mathbb{R}$*, es decir:*

$$L := \{3, 31, 314, 3141, 31415, \ldots\}$$

3.6. Minimización de autómatas deterministas

> Organisms [...] are directed and limited by their past. They must remain imperfect in their form and function, and to that extent unpredictable since they are not optimal machines
>
> Stephen Jay Gould

En ocasiones uno puede observar que el autómata que ha diseñado (usando algunas de las propiedades o métodos ya descritos) es un autómata con demasiados estados (y, por tanto, el código del programa es excesivo para el programador). Incluso aunque escribieramos un programa que generara el código para un autómata finito a partir de una especificación, siempre es mejor generar el autómata mínimo que uno que no lo sea. Por lo tanto, nos preguntamos si dado un autómata, podemos encontrar un autómata "con los menos estados posibles".
Para responder esta pregunta presentaremos un proceso de minimización de autómatas deterministas que pasaremos a describir a continuación. Este método se dividirá en varias fases, la primera de ellas es la eliminar los estados que no pueden ser alcanzados desde el estado inicial.

Eliminación de Estados Inaccesibles.

> Das Spiel dauert 90 Minuten.(los partidos duran 90 minutos)
>
> Sepp Hergerger

Esta frase ha sido elegida con un proposito muy especifico, mostrar que algunas informaciones no son necesarias y, por lo tanto no tiene sentido ni mencionarlas. Lo mismo ocurre con los autómatas finitos, empecemos por eliminar información sin cambiar el lenguaje que se acepta.

Es fácil dar ejemplos de autómatas deterministas en los que se han incluido estados innaccesibles, es decir, estados a los que no se puede llegar de ningún modo desde el estado inicial. Para describir esta noción, definiremos la siguiente estructura de grafo asociada a un autómata. Sea $A := (Q, \Sigma, \delta, q_0, F)$ un autómata determinista. Consideremos el grafo de estados siguiente: $\mathcal{G}_A := (V, E)$, donde

- El conjunto de vértices o nodos V es el conjunto de estados Q (i.e. $V := Q$).
- Las aristas del grafo (que será orientado) son los pares (q, q') tales que existe $a \in \Sigma$ verificando $\delta(q, a) = q'$.

Nótese que el grafo coincide con el grafo subyacente a la descripción del autómata como grafo con pesos.

Definición 64 *Dado un autómata $A := (Q, \Sigma, \delta, q_0, F)$, un estado $q \in Q$ se denomina accesible si está en la clausura transitiva (componente conexa) del estado inicial q_0. Se llaman estados inaccesibles aquellos estados que no son accesibles.*

Veremos que los autómatas solo necesitan los estados accesibles para ser definidos y que los estados inaccesibles son un lujo, en el peor sentido de la palabra.

Proposición 65 *Dado un autómata $A := (Q, \Sigma, \delta, q_0, F)$, existe un autómata B que acepta el mismo lenguaje y que no contiene estados inaccesibles.*

Demostración. Para definir B basta con eliminar los estados inaccesibles del autómata A, es decir, definimos $B := (Q', \Sigma, \delta', q_0, F')$ mediante

- $Q' := \{q \in Q \mid q \text{ es accesible desde } q_0 \text{ en } \mathcal{G}_A\}$.
- $F' := F \cap Q'$.
- La función de transición δ' es la restricción a Q' de δ.

El resto de la demostración, que la función de transición está bien definida y que el autómata B acepta el mismo lenguaje es un ejercicio que se deja para el alumno. ■

Autómata cociente

> An equal has no power over an equal
>
> Law Maxim

Seguimos nuestro camino para conseguir el mejor de los autómatas finitos que acepta un lenguaje. Este proceso comienza quitando aquellos estados que no aportaban nada al autómata, esto es, los estados inaccesibles. Ahora pasaremos a agrupar aquellos estados que dan el mismo resultado.

Definición 66 *Sea $A := (Q, \Sigma, \delta, q_0, F)$ un autómata determinista. Supongamos que todos los estados son accesibles. Dos estados $q, q' \in Q$ se dicen equivalentes si se verifica la siguiente propiedad:*

$$\forall x \in \Sigma^*, \text{ Si } \left(((q,x) \Rightarrow^* (q_1, \lambda)) \wedge \left((q', x) \Rightarrow^* (q_2, \lambda)\right)\right) \Longrightarrow ((q_1 \in F) \Leftrightarrow (q_2 \in F)).$$

En otras palabras, dos estados son equivalentes si para cualquier palabra, el efecto de la computación que generan es el mismo (en términos de alcanzar o no un estado final aceptador).

Denotaremos por $q \sim_A q'$ en el caso de que q y q' sean equivalentes. Para cada estado $q \in Q$, denotaremos por $[q]_A$ la clase de equivalencia definida por q y denotaremos por $Q/\sim_A$ al conjunto cociente. Definiremos autómata minimal al autómata que tiene el menor número de estados entre todos lo que aceptan un lenguaje.

Teorema 67 (Autómata cociente) *Sea L un lenguaje aceptado por un autómata determinista A sin estados inaccesibles. Entonces, existe un autómata minimal que lo acepta. Dicho autómata $(Q', \Sigma, q_0', F', \delta')$ viene dado en los términos siguientes:*

- $Q' := Q/\sim_A$,
- $F' := \{[q]_A \mid q \in F\}$.
- $q_0' := [q_0]_A$.
- $\delta'([q]_A, a) := [\delta(q, a)]$.

Demostración. Lo dejamos para la reflexión de los alumnos. ∎

El último teorema nos da un resultado muy valioso. Aunque sabemos que existe un autómata que es el minimal, en el sentido que tiene un número mínimo de estados, no sabemos todavía calcularlo. A este objetivo dedicaremos la siguiente sección.

Algoritmo para el cálculo de autómatas minimales.

> Los computadores son inútiles.
> Solo dan respuestas
>
> Pablo Picasso

De la definición del autómata cociente, concluimos la dificultad (aparente) del cálculo de las clases de equivalencia (porque habríamos de verificar todas las palabras $x \in \Sigma^*$). Por eso se plantean algoritmos alternativos como el que se describe a continuación (tomado de [Eilenberg, 74]).
Para construir nuestro autómata cociente, tomaremos una cadena de relaciones de equivalencia. Las definiremos recursivamente del modo siguiente:
Sea $A := (Q, \Sigma, \delta, q_0, F)$ un autómata. Definamos las siguientes relaciones:

- *La relación E_0:* dados $q, q' \in Q$, diremos que qE_0q' (q y q' están relacionados al nivel 0) si se verifica:

$$q \in F \iff q' \in F.$$

  Es claramente una relación de equivalencia. El conjunto cociente está formado por dos clases:

$$Q/E_0 := \{F, Q \setminus F\}.$$

  Definamos $e_0 := \sharp\,(Q/E_0) = 2$.

- *La relación E_1:* dados $q, q' \in Q$, diremos que qE_1q' (q y q' están relacionados al nivel 1) si se verifica:

$$qE_1q' \iff \begin{cases} qE_0q', \\ \wedge \\ \delta(q,a)E_0\delta(q',a), \quad \forall a \in \Sigma. \end{cases}$$

  Es, de nuevo, una relación de equivalencia. El conjunto cociente ya no es tan obvio, y definimos:

$$e_1 := \sharp\,(Q/E_1)\,.$$

- *La relación E_i:* para $i \geq 2$, definimos la relación del modo siguiente: Dados $q, q' \in Q$, diremos que qE_iq' (q y q' están relacionados al nivel i) si se verifica:

$$qE_iq' \iff \begin{cases} qE_{i-1}q', \\ \wedge \\ \delta(q,a)E_{i-1}\delta(q',a), \quad \forall a \in \Sigma. \end{cases}$$

  Es, de nuevo, una relación de equivalencia. El conjunto cociente ya no es tan obvio, y definimos:

$$e_i := \sharp\,(Q/E_i)\,.$$

Hemos definido una secuencia de enteros positivos $e_1, \ldots, e_i, \ldots$ tambien una serie de relaciones que dependen de i. Cada relación E_i es calculable. La idea es que para un valor i suficientemente grande, la relación que define E_i es exactamente igual a la dada por $\sim_A$.

Lema 68 *Sea $A := (Q, \Sigma, \delta, q_0, F)$ un autómata y sean $\{E_i \mid i \in \mathbb{N}\}$ la cadena de relaciones de equivalencia definidas conforme a la regla anterior. Se tiene:*

a. Para cada $i \in \mathbb{N}$, $e_i \leq e_{i+1}$.

b. Si existe $i \in \mathbb{N}$, tal que $e_i = e_{i+1}$, entonces

$$e_j = e_i,\ \forall j \geq i.$$

Demostración. Está claro que si dos estados están relacionados a nivel $i+1$ entonces, están relacionados a nivel i. Esto es así por como hemos contruido la relación. Por tanto, la relación E_{i+1} lo más que puede hacer es partir en más de una clase de equivalencia alguna de las clases de equivalencia del conjunto cociente anterior. Por tanto,

$$e_i = \sharp(Q/E_i) \leq \sharp(Q/E_{i+1}) = e_{i+1}.$$

Como, además, la relación E_{i+1} se define inductivamente a partir de la relación E_i, si $e_i = e_{i+1}$, entonces, las clases a nivel i siguen siendo las clases a nivel $i+1$. En otras palabras, si $e_i = e_{i+1}$, entonces para todo par $q, q' \in Q$, qE_iq' si y solamente si $qE_{i+1}q'$. En particular, $E_i = E_{i+1}$ y ambas relaciones de equivalencia son la misma. Inductivamente, para $i+2$ se tendrá

$$qE_{i+2}q' \iff \left\{ \begin{array}{c} qE_{i+1}q', \\ \\ \delta(q,a)E_{i+1}\delta(q',a), \quad \forall a \in \Sigma \end{array} \right\} \iff$$

$$\left\{ \begin{array}{c} qE_iq', \\ \\ \delta(q,a)E_i\delta(q',a), \quad \forall a \in \Sigma \end{array} \right\} \iff qE_{i+1}q' \iff qE_iq'.$$

Por tanto $E_{i+2} = E_{i+1} = E_i$ y, en consecuencia, $e_{i+2} = e_{i+1} = e_i$.
Para cualquier $j \geq i+3$, aplicando inducción para concluir $E_j = E_{i+1} = E_i$ y, además, $e_j = e_i$. Se dejan los detalles para el lector. ∎

Notar que estas relaciones son calculables algoritmicamente. La razón es que el alfabeto siempre es finito y que para cualquier autómata A, aunque sea indeterminista, su función de transición aplicada a cualquier estado y letra del alfabeto da un conjunto finito de estados. La siguiente proposición nos da una cota del número de relaciones que tenemos que estudiar antes de que la cadena de relaciones se estabilice.

Proposición 69 *Con las notaciones del lema anterior, para cada autómata A existe $i \in \mathbb{N}$, con $i \leq \sharp(Q) - 2$, tal que para todo $j \geq i$ se verifica:*

a. $qE_jq' \iff qE_iq'$, $\forall q, q' \in Q$.

b. $e_i = e_j$.

Demostración. Por el lema anterior, concluimos:

$$2 = e_0 \leq e_1 \leq e_2 \leq \cdots \leq e_i \leq \cdots$$

Ahora consideremos $i = \sharp(Q) - 2$. Pueden ocurrir dos cosas:

- *Caso I:* que $e_k \neq e_{k+1}$ para todo $k \leq i$. Es decir, se tiene (con $i = \sharp(Q) - 2$):

$$2 = e_0 < e_1 < e_2 < \cdots < e_i.$$

En este caso, tendríamos

$$\begin{aligned} e_1 \geq e_0 + 1 &= 3, \\ e_2 \geq e_1 + 1 &\geq 4, \\ &\vdots \\ e_{i-1} \geq e_{i-2} + 1 &\geq \sharp(Q), \\ \sharp(Q) \geq e_{i-1} &\geq \sharp(Q). \end{aligned}$$

Recordemos que $e_i = \sharp(Q/E_n)$ y el número de clases de equivalencia no puede ser mayor que el número de elementos de Q, es decir, habremos logrado que $i + 3 \leq e_i \leq \sharp(Q) = i + 2$ y eso es imposible.

- *Caso II:* la negación del caso anterior. Es decir, existe un k, con $0 \leq k \leq i$ (y $i = \sharp(Q) - 2$) tal que $e_k = e_k + 1$. Entonces, por el Lema anterior, se tendrá:

$$2 = e_0 < e_1 < e_2 < \cdots < e_k = e_{k+1} = \cdots = e_i = \cdots$$

Como el caso II es el único posible, es obvio que se tienen las propiedades del enunciado y esto acaba la demostración. ∎

Por fin, damos el teorema que explica claramente como conseguir el autómata minimal a partir de las relaciones E_i.

Teorema 70 *Sea $A := (Q, \Sigma, \delta, q_0, F)$ un autómata determinista y sean q, q' dos estados. Entonces, tomando $i = \sharp(Q) - 2$, se tendrá que*

$$q \sim_A q' \iff qE_iq'.$$

Demostración. Lo dejamos como ejercicio para el alumno. ∎

En particular, el algoritmo que calcula el autómata minimal funciona como sigue:

- *Hallar el conjunto cociente (Q/E_0) y su cardinal e_0.*
- *(Siguiendo los E_i's) Mientras el conjunto cociente "nuevo" sea alterado con respecto al anterior, hallar el conjunto cociente siguiente.*
- *Parar cuando el cardinal del "nuevo" conjunto cociente coincida con el último calculado.*

3.7. Cuestiones y problemas

Cuestiones

Cuestión 13 *Sea $A := (Q, \Sigma, q_0, F, \delta)$ un autómata indeterminista que verifica la siguiente propiedad: Para todo estado q y para todo símbolo $z \in \Sigma \cup \{\lambda\}$,*

$$\sharp\left(\{p \mid \delta(q, z) = p\}\right) \leq 1,$$

donde $\sharp$ significa cardinal. Dar un procedimiento inmediato para hallar uno equivalente que sea determinista.

Cuestión 14 *Hallar una expresión regular sobre el alfabeto* $\{a, b\}$ *que describa el lenguaje aceptado por el autómata siguiente. Sea* $Q := \{q_0, q_1\}$ *y* $F = \{q_1\}$*. Siendo la función de transición dada por la tabla:*

δ	**a**	**b**	λ
q_0	q_0	q_1	**N.D.**
q_1	**N.D.**	**N.D.**	**N.D.**

Donde **N.D.** *significa "No definido".*

Cuestión 15 *Considerar el autómata* $A := (Q, \Sigma, q_0, F, \delta)$*, donde*

- $\Sigma := \{a\}$,
- $Q := \{q_0, q_1, q_2\}$,
- $F := \{q_2\}$.

Y la función de transición es dada por la Tabla siguiente:

δ	**a**	λ
q_0	q_1	**N.D.**
q_1	q_2	**N.D.**
q_2	q_0	**N.D.**

Probar que $L(A) = \{(aaa)^n aa \mid n \in \mathbb{N}\}$.

Cuestión 16 *Describir un autómata que acepta el siguiente lenguaje:*

$$L(A) := \{\omega \in \{a, b\}^* \mid \sharp\,(\text{apariciones de } b \text{ en } \omega) \in 2\mathbb{N}\}.$$

Cuestión 17 *Considérese el autómata siguiente:*

- $\Sigma := \{0, 1\}$,
- $Q := \{q_0, q_1, q_2, q_3, q_4, q_5\}$,
- $F := \{q_2, q_3, q_4\}$.

Cuya función de transición es dada por la tabla siguiente:

δ	**0**	**1**	λ
q_0	q_1	q_2	**N.D.**
q_1	q_0	q_3	**N.D.**
q_2	q_4	q_5	**N.D.**
q_3	q_4	q_4	**N.D.**
q_4	q_4	q_5	**N.D.**
q_5	q_5	q_5	**N.D.**

a. Dibuja el grafo que describe al autómata.

b. Probar que q_0 y q_1 son equivalentes.

c. Probar que q_2, q_3, q_4 son equivalentes.

d. Hallar el autómata mínimo correspondiente.

Cuestión 18 *Sea G una gramática sobre el alfabeto $\{a, b\}$ cuyas reglas de producción son las siguientes:*

$$\begin{array}{rcl} S & \mapsto & bA \mid \lambda \\ A & \mapsto & bB \mid \lambda \\ B & \mapsto & aA \end{array}$$

Hallar una expresión regular que describa ese lenguaje. Hallar un autómata que acepte el lenguaje generado por esa gramática. Hallar el autómata mínimo que acepte ese lenguaje.

Cuestión 19 *Dado un autómata A que acepta el lenguaje L, ¿hay un autómata que acepta el lenguaje L^R?, ¿cómo le describirías?.*

Cuestión 20 *Dado un autómata A que acepta el lenguaje descrito por una expresión regular α y dado un símbolo a del alfabeto, ¿Cómo sería el autómata finito que acepta el lenguaje $L(D_a(\alpha))$?*

Problemas

Problema 33 *Construir autómatas que acepten los siguientes lenguajes:*

a. $L_1 := \{\omega \in \{a, b\}^* \mid abab \text{ es una subcadena de } \omega\}$.

b. $L_2 := \{\omega \in \{a, b\}^* \mid \text{ni } aa \text{ ni } bb \text{ son subcadenas de } \omega\}$.

c. $L_3 := \{\omega \in \{a, b\}^* \mid ab \text{ y } ba \text{ son subcadenas de } \omega\}$.

d. $L_4 := \{\omega \in \{a, b\}^* \mid bbb \text{ no es subcadena de } \omega\}$.

Problema 34 *Hallar un autómata determinista equivalente al autómata indeterminista $A := (Q, \{0, 1\}, q_0, F, \delta)$, donde*

- $\Sigma := \{0, 1\}$,
- $Q := \{q_0, q_1, q_2, q_3, q_4\}$,
- $F := \{q_4\}$.

Y δ es dado por la tabla siguiente:

δ	**a**	**b**	λ
q_0	**N.D.**	q_2	q_1
q_1	q_0, q_4	**N.D.**	q_2, q_3
q_2	**N.D.**	q_4	**N.D.**
q_3	q_4	**N.D.**	**N.D.**
q_4	**N.D.**	**N.D.**	q_3

Problema 35 *Minimizar el autómata siguiente:*

- $\Sigma := \{0, 1\}$,
- $Q := \{q_0, q_1, q_2, q_3, q_4, q_5\}$,
- $F := \{q_3, q_4\}$.
- δ *esta dada por la siguiente tabla:*

δ	**0**	**1**	λ
q_0	q_1	q_2	**N.D.**
q_1	q_2	q_3	**N.D.**
q_2	q_2	q_4	**N.D.**
q_3	q_3	q_3	**N.D.**
q_4	q_4	q_4	**N.D.**
q_5	q_5	q_4	**N.D.**

Hallar su grafo, una gramática que genere el mismo lenguaje y una expresión regular que lo describa.

Problema 36 *Construir una expresión regular y un autómata finito asociados al lenguaje siguiente:*

$$L := \{\omega \in \{a, b\}^* \mid \exists z \in \{a, b\}^*,\ \omega = azb\}.$$

Problema 37 *Hallar una expresión regular y una gramática asociadas al lenguaje aceptado por el autómata* $A := (Q, \Sigma, q_0, F, \delta)$*, dado por las propiedades siguientes*

- $\Sigma := \{a, b\}$,
- $Q := \{q_0, q_1, q_2, q_3, q_4\}$,
- $F := \{q_3, q_4\}$.

Y δ *es dado por la tabla siguiente:*

δ	**a**	**b**	λ
q_0	q_1	**N.D.**	**N.D.**
q_1	q_2	q_4	**N.D.**
q_2	q_3	q_4	**N.D.**
q_3	q_3	q_4	**N.D.**
q_4	**N.D.**	q_4	**N.D.**

Problema 38 *Hallar un autómata determinista que acepta el lenguaje descrito por la siguiente expresión regular:*

$$a(bc)^*(b + bc) + a.$$

Minimiza el resultado obtenido.

Problema 39 *Haz lo mismo que en el problema anterior para la expresión regular:*

$$(a\,(ab)^*)^*\,da\,(ab)^*\,.$$

Problema 40 *Haz lo mismo que en el problema anterior para la expresión regular:*

$$0(011)^*0 + 10^*\,(1\,(11)^*\,0 + \lambda) + \lambda.$$

Problema 41 *Calcula un autómata finito determinista minimal, una gramática regular y una expresión regular para el lenguaje siguiente:*

$$L := \{\omega \in \{0,1\}^* \ : \ [\sharp\,(0\text{'s en } \omega) \in 2\mathbb{N}] \vee [\sharp\,(1\text{'s en } \omega) \in 3\mathbb{N}]\}.$$

Problema 42 *Obtener una expresión regular para el lenguaje aceptado por* $A := (Q, \Sigma, q_0, F, \delta)$,

- $\Sigma := \{a, b\}$,
- $Q := \{q_0, q_1, q_2\}$,
- $F := \{q_2\}$.

Y δ *es dado por la tabla siguiente:*

δ	**a**	**b**	λ
q_0	q_0, q_2	**N.D.**	q_1
q_1	q_2	q_1	**N.D.**
q_2	**N.D.**	**N.D.**	q_1

Problema 43 *Dada la expresión regular* $(ab)^*(ba)^* + aa^*$, *hallar:*

a. *El autómata determinista minimal que acepta el lenguaje que describe esa expresión regular.*

b. *Una gramática regular que genere dicho lenguaje.*

Problema 44 *Considera un tipo de datos* real *de algún lenguaje de programación. Halla una expresión regular que describa este lenguaje. Halla un autómata que los reconozca y una gramática que los genere.*

Problema 45 *Se considera el autómata descrito por la información siguiente* $A := (Q, \Sigma, q_0, F, \delta)$, *y*

- $\Sigma := \{0, 1\}$,
- $Q := \{q_0, q_1, q_2, q_3, q_4, q_5\}$,
- $F := \{q_1, q_3, q_5\}$.

Y δ *es dado por la tabla siguiente:*

δ	**0**	**1**	λ
q_0	**N.D.**	**N.D.**	q_1
q_1	q_2	**N.D.**	**N.D.**
q_2	**N.D.**	q_3	**N.D.**
q_3	q_4	**N.D.**	q_1
q_4	q_5	**N.D.**	**N.D.**
q_5	**N.D.**	**N.D.**	q_3, q_1

Se pide:

a. *Dibujar el grafo de transición del autómata.*

b. *Decidir si* 0101 *es aceptado por al autómata y describir la computación sobre esta palabra.*

c. *Hallar un autómata determinista que acepte el mismo lenguaje.*

d. *Minimizar el autómata determinista hallado.*

e. *Hallar una gramática que genere dicho lenguaje.*

f. *Hallar una expresión regular que describa el lenguaje aceptado por ese autómata.*

Problema 46 *Hallar un autómata que acepte las expresiones matemáticas con sumas y restas y sin paréntesis. Añadir a este una cinta donde escribir la traducción al español acabando en punto. Ejemplo:*

$$4 + 3 - 5 \mapsto \textit{ cuatro mas tres menos cinco.}$$

Problema 47 *Suponer que al autómata anterior se le quisiera añadir expresiones con paréntesis. Para hacer esto toma la expresión regular del autómata anterior α y se considera la siguiente expresión regular $(^*\alpha)^*$. Demostrar que el autómata no comprueba que todos los paréntesis que se abren son cerrados.*

Problema 48 *Otro de los problemas de los autómatas finitos es que no tienen en cuenta el orden entre los distintos elementos. Utilicemos una expresión regular α mencionada en el ejercicio anterior. Hallar el autómata que acepte el lenguaje generado por la siguiente expresión regular*

$$('('+'\{')^* alpha(')'+'\}')^*.$$

Demostrar que el autómata no tiene en cuenta el orden de aparición de las llaves y los paréntesis.

Problema 49 *Este ejercicio demuestra el problema de traducción para las estructuras condicionales. Suponemos que EXPRESION es conocido y los bucles condicionales están dados por la siguiente expresión regular:*
((if EXPRESION then BUCLE) (else if EXPRESION then BUCLE)) Hallar un autómata finito que acepte el lenguaje dado por la expresión regular y discutir como añadir una cinta de traducción.*

Capítulo 4

Gramáticas libres de contexto

Índice

4.1. Introducción

Hemos comentado varias veces que estos apuntes tienen como objetivo inculcar al lector con una base teórica con la cual abordar otros problemas. El proceso de compilación es el proceso que justifica la presencia de un estudio teórico de lenguajes y autómatas como el que se desarrolla en esta asignatura. Sin embargo, el objetivo de un curso como éste no es, ni debe ser, el del diseño de un compilador, ni siquiera el del análisis de todos los procesos que intervienen en la compilación. Para culminar este proceso, existen varios buenos libros, como "Compilers: Principles, techniques and tools" de A. V. Aho, M. S. Lam, R. Sethi y J. D. Ullman.
Volvemos a repetir que esperamos despertar el interes del lector porque esta asignatura supone un reto y formaliza las bases de una de las areas mejor entendidas de las ciencias de computación.
Por ello, en estos apuntes seguimos el esquema básico de un clásico como los dos volúmenes (que han influido intensamente en el diseño del presente manuscrito) como [Aho-Ullman, 72a] y [Aho-Ullman, 72b].
Por conveniencia al lector, recordemos que una Gramática Libre de Contexto (CFG) o de Tipo 2 es dada por la siguiente definición.

Definición 71 (Gramáticas libres de contexto o de Tipo 2) *Llamaremos gramática libre de contexto a toda gramática* $G := (V, \Sigma, S, \mathcal{P})$ *tal que todas las producciones de* $\mathcal{P}$ *son del tipo siguiente:*

$$A \to w, \text{ donde } A \in V \text{ y } w \in (\Sigma \cup V)^*.$$

Un lenguaje **libre de contexto** *es un lenguaje generado por una gramática libre de contexto.*

En el presente capítulo nos ocupamos de resolver el Problema de la palabra para gramáticas libres de contexto y mostraremos cómo reducir a formas normales las gramáticas libres de contexto, lo que simplificará el análisis de la equivalencia con los PDA's.

4.2. Árboles de derivación de una gramática

En un sistema de transición dos configuraciones puedes ser deducibles por muchas diferentes computaciones. De la misma forma, hay muchas diferentes derivaciones para una misma palabra. Una de nuestras primeras tareas será buscar una expresión de todas las derivaciones.
Para poder de hablar de las derivaciones, es necesario introducir los siguientes conceptos clásicos que serán muy útiles a la hora de definir los estados intermedios.

Definición 72 (Formas sentenciales y formas terminales) *Llamamos formas sentenciales a todos los elementos* x *de* $(V \cup \Sigma)^*$. *Llamaremos formas terminales a las formas sentenciales que sólo tienen símbolos en el alfabeto de símbolos terminales, es decir, a los elementos de* Σ^*.

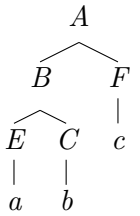
Ahora estamos en disposición de definir el concepto *árbol de derivación*, que codificará varias derivaciones en un solo objeto.

Definición 73 (Árbol de Derivación) *Sea* $G := (V, \Sigma, S, \mathcal{P})$ *una gramática libre de contexto, sea* $A \in V$ *una variable. Diremos que un árbol* $\mathcal{T}_A = (\mathcal{V}, \mathcal{E})$ *etiquetado es un árbol de derivación asociado a* G *si verifica las propiedades siguientes:*

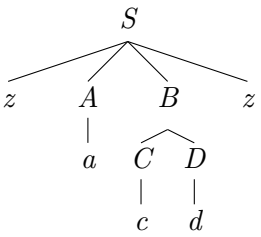
- *La raíz del árbol es el símbolo no terminal* A.
- *Las etiquetas de los nodos del árbol son símbolos en* $V \cup \Sigma \cup \{\lambda\}$.
- *Cada nodo interior está etiquetado con un símbolo en* V *(i.e. un símbolo no terminal).*
- *Cada hoja está etiquetada con un símbolo terminal o* λ.
- *Si un nodo está etiquetado con una variable* B *y las etiquetas de sus descendientes (leídos de izquierda a derecha) en el árbol forman la palabra* $x \in (V \cup \Sigma)^*$ *entonces, hay una producción* $B \to x$ *en* $\mathcal{P}$.

Ejemplo 10 *Hallar árboles de derivación para las gramáticas siguientes:*

- *$A \to BF, B \to EC, E \to a, C \to b, F \to c$, de tal manera que la raíz sea A y las hojas estén etiquetadas con a, b, c en este orden.*



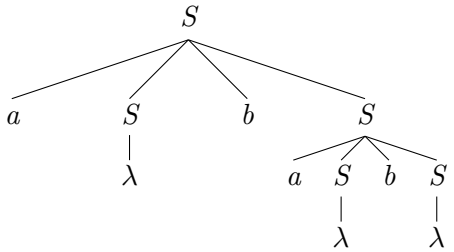
- *$S \mapsto zABz, B \mapsto CD, C \mapsto c, D \mapsto d, A \mapsto a$, de tal manera que la raíz sea S y las hojas estén etiquetadas (de izquierda a derecha) mediante z, a, c, d, z.*



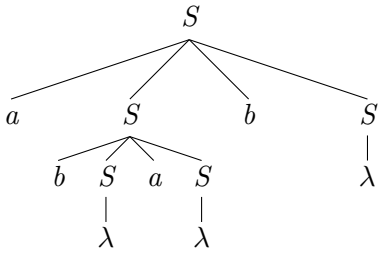
- *$S \mapsto aSbS \mid bSaS \mid \lambda$. Escribe árboles de derivación cuyas hojas tengan la lectura siguiente:*

  - *Una sóla hoja con λ.*

  - *Un árbol con varias hojas, tales que leyendo la palabra se obtenga abab.*



  - *Un árbol distinto, con varias hojas tales que leyendo la palabra se obtenga abab.*



Estos ejemplos ilustran una relación entre árboles de derivación y derivaciones. Esta relación la expresaremos en la siguiente proposición.

Proposición 74 *Sea $G := (V, \Sigma, S, \mathcal{P})$ una gramática libre de contexto y sea $A \in V$ una variable. Sea $\mathcal{T}_A$ un árbol asociado a la gramática con raíz A y $x \in (V \cup \Sigma)^*$ la forma sentencial obtenida leyendo de izquierda a derecha los símbolos de las hojas de $\mathcal{T}_A$. Entonces, $A \to^* x$. En particular, las formas sentenciales alcanzables desde el símbolo inicial S está representados por los árboles de derivación de S.*

Demostración. Obvio a partir de la definición. ∎

Notar que las palabras que son generadas por una gramática son las que son deducibles desde la variable inicial. Notando esto, es trivial deducir el siguiente corolario.

Corolario 75 *Las palabras sobre el alfabeto Σ están en el lenguaje $L(G)$ generado por una gramática G si y solamente si existe un árbol de derivación cuyas hojas (leídas de izquierda a derecha) expresan la palabra palabra.*

4.3. Un algoritmo incremental para la vacuidad

> En la historia interminable II se luchaba contra la nada, el vacio. Una tonteria. Pero claro, despues de la primera parte, cualquiera le decia a Michael Ende que hiciera un nuevo guión
>
> Conversación privada

Comenzaremos con un ejemplo de un algoritmo que será reutilizado, por analogía en los demás algoritmos de esta subsección. Su objetivo es decidir si es o no vacío el lenguaje generado por una gramática libre de contexto.

Teorema 76 (Vacuidad de un lenguaje libre de contexto) *El problema de la vacuidad de los lenguajes generados por gramáticas libres de contexto es decidible. Es decir, existe un algoritmo que toma como input una gramática libre de contexto G y devuelve una respuesta afirmativa si $L(G) \neq \emptyset$ y negativa en caso contrario.*

Demostración. Definimos el algoritmo siguiente:
Obsérvese que este algoritmo tiene la propiedad de que los sucesivos conjuntos M y N construidos en su recorrido son siempre subcojuntos del conjunto de símbolos no terminales V. Por tanto, es un algoritmo. En otras palabras, es un procedimiento que acaba en todos los datos de entrada.
Veamos que este algoritmo realiza las tareas prescritas. Para ello, consideremos una cadena de subconjuntos N_i de V que reflejan los sucesivos pasos por el ciclo while.

Algorithm 6 Algoritmo para testear si una gramática genera al menos una palabra

Input: Una gramática $G := (V, \Sigma, S, \mathcal{P})$
Output: Verdadero si la palabra genera al menos una palabra y falso en caso contrario

```
M = ∅
N = {A ∈ V | A → x ∈ P, x ∈ Σ*}
while M ≠ N do
    M = N
    N = {A ∈ V | A → x ∈ P, x ∈ (Σ ∪ M)*}
end while
if S ∈ M then
    return Verdadero
else
    return Falso
end if
```

Escribamos $N_0 = \emptyset$ y denotemos por N_i al conjunto obtenido en el $i-$ésimo paso por el ciclo while, sin considerar la condición de parada. Esto es,

$$N_i := \{A \in V \mid A \to x \in \mathcal{P},\ x \in (\Sigma \cup N_{i-1})^*\}.$$

Está claro que tenemos una cadena ascendente

$$N_0 \subseteq N_1 \subseteq N_2 \subseteq \cdots \subseteq N_n \subseteq \cdots \subset V.$$

Por construción observamos, además, que si existe un paso i tal que $N_i = N_{i+1}$, entonces, $N_i = N_m$ para todo $m \geq i + 1$.
Analicemos qué propiedades han de verificar las variables en N_i. Por inducción se probará lo siguiente:

Una variable $V' \in V$ verifica que $V' \in N_i$ si y solamente si existe un árbol de derivación de G de altura[1] *$i+1$ que tiene A como raíz y cuyas hojas están etiquetadas con símbolos en $\Sigma \cup \{\lambda\}$.*

Una vez probada esta propiedad, está claro que se tiene:

- Sea i tal que nuestro algoritmo detiene sus cálculos tras i pasos por el ciclo while. Sea N el conjunto de variables calculado en ese paso. Entonces, $N = N_m$, $\forall m \geq i + 1$.
- Si $S \in N$, entonces, $N = N_{i+1}$ y existe un árbol de derivación de la gramática de altura $i + 2$ cuyas hojas son todo símbolos en $\Sigma \cup \{\lambda\}$ y cuya raíz es S. Sea $x \in \Sigma^*$, la palabra descrita mediante la lectura (de izquierda a derecha) de las hojas del árbol. Entonces, $S \to^* x \in \Sigma^*$, luego $x \in L(G)$.
- Por otro lado, tomando cualquier palabra $x \in L(G) \neq \emptyset$ habrá un árbol de derivación cuya raíz es S y sus hojas están etiquetadas con elementos de

[1] Medimos altura por el número de nodos atravesados en el camino más largo.

$\Sigma \cup \{\lambda\}$. Además leyendo de izquierda a derecha las hojas obtenemos x. Sea m la altura de tal árbol ($m \geq 1$, obviamente) y, por tanto, $S \in N_{m-1} \subseteq N_{i+1} = N$ para cualquier m.

Esto acaba la demostración. ∎

Ahora sabemos como testear si una gramática genera al menos una palabra.

4.4. Algoritmos para gramáticas libres de contexto

> Han creado algo maravilloso, es como el que recubre una pasa de chocolate o viste un mono de cowboy
>
> Homer J. Simpson

El objetivo de esta sección es realizar una serie de reducciones algorítmicas (transformaciones de gramáticas) hasta reducir una gramática libre de contexto a una gramática en Forma Normal de Chomsky. Las diferentes subsecciones están basadas en las progresivas reducciones y simplificaciones.

Definición 77 *Dos gramáticas libres de contexto G y G_1 se dicen equivalentes, si generan el mismo lenguajes, esto es, si $L(G) = L(G_1)$.*

De ahora en adelante, todas las transformaciones de gramáticas serán transformaciones que preserven la condición de "ser equivalentes". La primera observación que vamos a hacer es que hay muchas variables que pueden ser eliminadas porque no se utilizan para generar palabras. En la siguiente definición, vamos a dar una descripción formal de las variables que podemos eliminar.

Definición 78 (Símbolos Inútiles) *Sea $G := (V, \Sigma, S, \mathcal{P})$ una gramática libre de contexto. Llamamos símbolos útiles de G a todos los símbolos (terminales o no) $A \in V \cup \Sigma$ tales que existen $x, y \in (V \cup \Sigma)^*$ y $w \in \Sigma^*$ de tal modo que:*

$$S \to^* xSy, \text{ y } xSy \to^* w.$$

Los símbolos inútiles son los que no son útiles.

Ejemplo 11 *Consideremos la gramática $G := (\{S, A, B\}, \{a, b\}, S, \mathcal{P})$, donde las producciones de $\mathcal{P}$ son las siguientes:*

$$\mathcal{P} := \{S \to a \mid A,\ A \to AB,\ B \to b\}.$$

Obsérvese que b, A, B son símbolos inútiles en esta gramática. La razón es que el símbolo A no produce ningún símbolo, por tanto el lenguaje aceptado es $\{a\}$ ya que eliminar símbolos inútiles de una gramática da una gramática equivalente. Si, por el contrario, añadiéramos la producción $A \to a$, entonces, todos ellos serían símbolos útiles.

Definición 79 *Sea $G := (V, \Sigma, S, \mathcal{P})$ una gramática libre de contexto. Se tiene lo siguiente:*

- *Llamamos símbolos productivos (o fecundos) de G a todos los símbolos no terminales $A \in V$ tales que existe $x \in \Sigma^*$ tal que $A \to^* x$. Son improductivos (o infecundos) a los que no satisfacen esta propiedad.*
- *Llamamos símbolos accesibles de G a todos los símbolos (terminales o no) $A \in V' \cup \Sigma$ tales que existen $x, y \in (V \cup \Sigma)^*$ de tal modo que:*

$$S \to^* xAy.$$

Se llaman inaccesibles a los que no son accesibles.

Ejemplo 12 *Nótese que si A es un símbolo útil, se han de producir dos propiedades. De una parte, la propiedad $S \to^* xAy$ que nos dice que A es accesible. De otra parte, por estar en una gramática libre de contexto, ha de existir $w \in \Sigma^*$ tal que $A \to^* w$. Esto es necesario porque, al ser libre de contexto, todas las producciones se basan en reemplazar una variables por formas sentenciales. Si la variable A no alcanzará nunca una forma terminal en el sistema de transición, entonces, xAy tampoco alcanzaría una forma terminal contradiciendo el hecho de ser A útil. La existencia de $w \in \Sigma^*$ tal que $A \to^* w$ nos dice que A es un símbolo fecundo o productivo. En el siguiente ejemplo:*

$$\mathcal{P} := \{S \to AB \mid CD, A \to AS, B \to b, C \to Cb \mid \lambda, D \to b\}.$$

El símbolo B es fecundo y accesible, pero es un símbolo inútil.

Proposición 80 *Si $G := (V, \Sigma, S, \mathcal{P})$ es una gramática libre de contexto, entonces los símbolos útiles son productivos y accesibles. El recíproco no es cierto.*

Demostración. Es obvia la implicación enunciada. En cuanto a ejemplos que muestran que el recíproco no es en general cierto, baste con ver las gramáticas expuestas en los Ejemplos 11 y 12. ∎

Proposición 81 *Si $G := (V, \Sigma, S, \mathcal{P})$ es una gramática libre de contexto, y libre de símbolos infecundos, entonces todo símbolo es útil si y solamente si es accesible.*

Demostración. En ausencia de símbolos infecundos accesibilidad es sinónimo de utilidad. La prueba es la obvia. ∎

Proposición 82 (Eliminación de símbolos infecundos) *Toda gramática libre de contexto es equivalente a una gramática libre de contexto sin símbolos infecundos. Además, dicha equivalencia puede hacerse de manera algorítmica.*

Demostración. El algoritmo 7 es esencialmente el propuesto en el Teorema 76 anterior:
Por la prueba del Teorema 76, sabemos que N es justamente el conjunto de variables productivas y el algoritmo realiza la tarea pretendida. ∎

Algorithm 7 Algoritmo para eliminar símbolos improductivos

Input: Una gramática $G := (V, \Sigma, S, \mathcal{P})$
Output: Una gramática equivalente sin símbolos improductivos

$M := \emptyset$
$N := \{A \in V \mid A \to x \in \mathcal{P},\ x \in \Sigma^*\}$
while $M \neq N$ **do**
  $M := N$
  $N := \{A \in V \mid A \to x \in \mathcal{P},\ x \in (\Sigma \cup M)^*\}$
end while
if $S \notin M$ **then**
  return $(\{S\}, \Sigma, S, \emptyset)$
else
  return $(N, \Sigma, S, \mathcal{P}')$ donde $\mathcal{P}'$ son las producciones de $\mathcal{P}$ que involucran solamente símbolos en $(N) \cup \Sigma \cup \{\lambda\}$
end if

Teorema 83 (Eliminación de símbolos inaccesibles) *Toda gramática libre de contexto es equivalente a una gramática libre de contexto sin símbolos inaccesibles. Además, dicha equivalencia puede hacerse de manera algorítmica.*

Demostración. El algoritmo 8 elimina símbolos inaccesibles de una gramática libre de contexto. La demostración de que es un algoritmo y de que realiza la tarea prevista es análoga a la demostración del Teorema 76 anterior. Nótese que, de facto, el algoritmo calcula los símbolos que sí son accesibles. ∎

Algorithm 8 Algoritmo para eliminar símbolos inaccesibles

Input: Una gramática $G := (V, \Sigma, S, \mathcal{P})$
Output: Una gramática equivalente sin símbolos inaccesibles

$M := \emptyset$
$N := \{A \in V \mid S \to xAy \in \mathcal{P},\ x, y \in (\Sigma \cup V)^*\}$
while $M \neq N$ **do**
  $M := N$
  $N := \{A \in V \mid \exists B \in M,\ B \to xAy \in \mathcal{P},\ x, y \in (\Sigma \cup M)^*\}$
end while
return $(N, \Sigma, S, \mathcal{P}')$ donde $\mathcal{P}'$ son las producciones de $\mathcal{P}$ que involucran solamente símbolos en $(V \cap N) \cup \Sigma \cup \{\lambda\}$

Teorema 84 (Eliminación de símbolos inútiles) *Toda gramática libre de contexto es equivalente a una gramática sin símbolos inútiles. Además, esta equivalencia es calculable algorítmicamente.*

Demostración. El algoritmo que combina los dos algoritmos descritos anteriormente y el enunciado de la Proposición 81 permite eliminar los símbolos inútiles. En resumen, primero hay que eliminar los símbolos infecundos con sus producciones y luego los símbolos inaccesibles. ∎

Seguiremos con un paso esencial para encontrar una familia de gramáticas entre todas las gramáticas equivalentes a una dada. Estas gramáticas son llamadas *gramáticas propias*. Veremos como están gramáticas son construibles de una forma eficiente y facilmente programable. En nuestro camino hacia este conocimiento, continuaremos con transformaciones de las gramáticas libres de contexto hasta obtener gramáticas propias.

Definición 85 *Sea $G := (V, \Sigma, S, \mathcal{P})$ una gramática libre de contexto.*

a. *Llamaremos $\lambda-$producciones en G a todas las producciones de la forma $A \to \lambda$, donde $A \in V$ es un símbolo no terminal.*

b. *Diremos que la gramática G es $\lambda-$libre si verifica una de las dos propiedades siguientes:*

   - *no posee $\lambda-$producciones,*
   - *o bien la única $\lambda-$producción es de la forma $S \to \lambda$ y S no aparece en el lado derecho de ninguna otra producción de $\mathcal{P}$ (es decir, no existe ninguna producción de la forma $S \to xSy$, con $x, y \in (V \cup \Sigma)^*$).*

Ejemplo 13 *Consideremos la gramática cuyas producciones son:*

$$S \to aSbS \mid bSaS \mid \lambda.$$

No es una gramática $\lambda-$libre.

En el siguiente teorema, vemos como realizar una transformación de una gramática cualquiera a una gramática λ-libre.

Teorema 86 (Transformación a gramática $\lambda-$libre) *Toda gramática libre de contexto es equivalente a una gramática $\lambda-$libre. Además, dicha equivalencia es calculable algorítmicamente.*

Demostración. El algoritmo comienza con una tarea que repite esencialmente lo hecho en algoritmos anteriores. Se trata de hacer desaparecer las variables que van a parar a la palabra vacía λ; pero de manera selectiva. No las eliminamos completamente porque podrían ir a parar a constantes o formas terminales no vacías.
Hallar $V_\lambda := \{A \in V \ : \ A \to^* \lambda\}$.
A partir del cálulo de V_λ procedemos de la forma siguiente:

a. Calculamos el nuevo sistema de producciones $\overline{\mathcal{P}}$ del modo siguiente:

   - Consideremos todas las producciones de la forma siguiente:

$$A \to \alpha_0 B_1 \alpha_1 \cdots B_k \alpha_k,$$

   donde $\alpha_i \notin V_\lambda^*$, $B_i \in V_\lambda$ y no todos los α_i son iguales a λ. Definamos

$$\overline{\mathcal{P}} := \overline{\mathcal{P}} \cup \{A \to \alpha_0 C_1 \alpha_1 \cdots C_k \alpha_k \ : \ C_i \in \{B_i, \lambda\}\}.$$

- Consideremos todas las producciones de la forma siguiente:

$$A \to B_1 \cdots B_k,$$

donde $B_i \in V_\lambda$. Definamos:

$$\overline{\mathcal{P}} := \overline{\mathcal{P}} \cup (\{A \to C_1 \cdots C_k \ : \ C_i \in \{B_i, \lambda\}\} \setminus \{A \to \lambda\}) .$$

b. Eliminamos todas las $\lambda-$producciones restantes.

c. Finalmente, si $S \in V_\lambda$ sea $\overline{V} = V \cup \{S_1\}$, con $S_1 \notin V$. Y añadamos

$$\overline{\mathcal{P}} := \overline{\mathcal{P}} \cup \{S_1 \to S \mid \lambda\}.$$

En otro caso, $\overline{V} := V$.

El output será la gramática $\overline{G} := (\overline{V}, \Sigma, S_1, \overline{\mathcal{P}})$ y satisface las propiedades de las hipótesis. ∎

Nota 87 *La eliminación de $\lambda-$producciones puede tener un coste exponencial en el máximo de las longitudes de las formas sentenciales (en $(\Sigma \cup V)^*$) que aparecen a la derecha de las producciones de la gramática dada.*

Aunque seguramente el lector estará preguntandose porque se consideran tan importantes las λ-producciones, la respuesta solo llegará despues de varias páginas. Rogamos una vez más la paciencia del lector, porque creemos que merecerá la pena.

Definición 88 (Producciones simples o unarias) *Se llaman producciones simples (o unarias) a las producciones de una gramática libre de contexto de la forma $A \mapsto B$, donde A y B son símbolos no terminales.*

Teorema 89 (Eliminación de producciones simples) *Toda gramática $\lambda-$libre es equivalente a una gramática $\lambda-$libre y sin producciones simples. Esta equivalencia es calculable algorítmicamente.*

Demostración. El algoritmo tiene dos partes. La primera parte sigue el mismo esquema algorítmico usado en resultados anteriores. La segunda parte se dedica a eliminar todas las producciones simples.

- *Clausura transitiva de símbolos no terminales.* Se trata de calcular, para cada $A \in V$, el conjunto siguiente:

$$N_A = \{B \in V \mid A \to^* B\} \cup \{A\}.$$

Nótese que se trata de la clausura transitiva en el grafo $(V, \to)$, inducido por el sistema de transición sobre el conjunto de variables. El algoritmo 9 es la expresión de la forma de resolver el problema más directa:

Algorithm 9 Algoritmo para calcular la clausura transitiva de una variable

Input: Una gramática $G := (V, \Sigma, S, \mathcal{P})$ y $\lambda-$libre
Output: Una gramática equivalente sin producciones unarias
  for $A \in V$ **do**
    $M_A := \emptyset$
    $N_A := \{A\}$
    while $M_A \neq N_A$ **do**
      $M_A := N_A$
      $N_A := \{C \in V \mid \exists B \in M_A,\ B \to C \in \mathcal{P}\}$
    end while
  end for
  return N_A for $A \in V$

- También podemos definir el conjunto de los antepasados de una variable

$$V_A := \{B \in V \mid B \Rightarrow^* A\} = \{B \in V \mid A \in N_B\}.$$

  Es calculable por una modificación del algoritmo anterior de una forma obvia.

- *Eliminar las producciones simples.* Para cada variable B tal que existe una producción simple $A \mapsto B$ en $\mathcal{P}$,

  - Hallar todos los $B \in N_C$.
  - Para cada producción $B \mapsto x$ que no sea producción simple, añadir a $\mathcal{P}$ la producción $C \mapsto x$.
  - Eliminar todas las producciones del tipo $C \mapsto B$.

Nótese que cada iteración de la parte segunda del proceso añade producciones no unitarias y elimina al menos una producción simple. Con ello se alcanza el objetivo buscado. ■

La necesidad de eliminar producciones unarias y $\lambda-$producciones viene por la presencia de ciclos a la hora de resolver el problema de la palabra. En teoría, si aplicamos diferentes reglas gramaticales y no hallamos ninguna derivación de la palabra no podemos descartar que la palabra está generada por la gramática o no. En el caso de gramáticas "aciclicas" veremos como demostrar, después de probar suficientes combinaciones de reglas gramaticales, si una palabra está generada por la gramática o no.

Definición 90 *Diremos que una gramática libre de contexto $G := (V, \Sigma, S, \mathcal{P})$ es acíclica (o libre de ciclos) si no existe ningún símbolo no terminal $A \in V$ tal que existe una computación no trivial (en el sistema de transición asociado):*

$$A \to x_1 \to \cdots \to x_k = A.$$

La siguiente definición trata sobre las gramáticas que nos interesarán.

Definición 91 (Gramáticas propias) *Diremos que una gramática libre de contexto $G := (V, \Sigma, S, \mathcal{P})$ es propia si verifica las siguientes propiedades:*

- *G es acíclica,*
- *G es $\lambda-$libre,*
- *G es libre de símbolos inútiles.*

Lema 92 *[Interacción entre los algoritmos expuestos] Se dan las siguientes propiedades:*

a. *Si G es una gramática libre de contexto que es $\lambda-$libre y está libre de producciones simples, entonces G es acíclica.*

b. *Sea G una gramática libre de contexto, $\lambda-$libre. Sea G_1 la gramática obtenida después de aplicar a G el algoritmo de eliminación de producciones simples descrito en la demostración del Teorema 89. Entonces, G_1 sigue siendo $\lambda-$libre.*

c. *Sea G una gramática libre de contexto, libre de producciones simples y $\lambda-$libre. Sea G_1 la gramática obtenida después de aplicar a G el algoritmo de eliminación de símbolos inútiles descrito en la demostración del Teorema 84. Entonces, G_1 sigue siendo libre de producciones simples y $\lambda-$libre.*

Demostración.

a. Supongamos que la gramática fuera $\lambda-$libre y libre de producciones simples, pero hubiera un estado que generara un ciclo. Es decir, supongamos que existe:

$$A \to x_1 \to \cdots \to x_k \to A,$$

con $k \geq 1$. Entonces, puedo suponer que

$$x_k := \alpha_0 C_1 \alpha_1 \cdots \alpha_{n-1} C_n \alpha_n,$$

donde $\alpha_i \in \Sigma^*$, $C_i \in V$. En primer lugar, obsérvese que, como estamos hablando de gramáticas libres de contexto, las únicas producciones que se aplican son de la forma $B \to y$. Si alguno de los α_i fuera distinto de la palabra vacía λ, no habría forma de "borrarlos" usando las producciones libre de contexto (para quedarnos solamente con un símbolo no terminal como A). Por tanto, $\alpha_i = \lambda$ para cada i, $0 \leq i \leq n$. En conclusión, sólo tenemos (como última acción):

$$x_k = C_1 \cdots C_n \to A.$$

Si, además, $n \geq 2$, con una única producción libre de contexto, no podríamos eliminar nada más que un símbolo no terminal. Con lo cual no podríamos obtener A. Por tanto, tenemos, en realidad, $n \leq 2$. Tenemos dos casos $n = 1$ o $n = 2$. Si $n = 1$ tendremos:

$$x_k = C_1 \to A.$$

En este caso debería existir la producción simple $C_1 \to A$, pero hemos dicho que nuestra gramática es libre de producciones simples. Por tanto, $n = 1$ no puede darse. Nos queda un único caso $n = 2$ y tendremos:

$$x_k = C_1C_2 \to A.$$

En este caso hay dos opciones simétricas, con lo que discutiremos sólo una de ellas: $C_1 = A$ y $C_2 \to \lambda$ es una producción. Pero nuestra gramática es $\lambda-$libre. Si hay una producción $C_2 \to \lambda$ es sólo porque C_2 es el símbolo inicial $C_2 = S$. Por tanto, tenemos una computación:

$$A \to x_1 \to \cdots \to x_{k-1} \to AS \to A,$$

Pero, para llegar a AS desde A tiene que ocurrir que S esté en la parte derecha de alguna producción (lo cual es imposible por la propia defínción de $\lambda-$libre). Llegamos a una contradicción, por lo tanto no puede haber ciclos.

b. Retomemos el algoritmo descrito en la prueba del Teorema 89. Una vez calculado V_A para cada A, tratamos las producciones unarias del tipo $A \to B$ del modo siguiente:

   - Hallar todos las C's tales que $B \in N_C$.
   - Para cada producción $B \to x$ donde $x \in (V \cup \Sigma)^+$ que no sea producción simple, añadir a $\mathcal{P}$ la producción $C \to x$.
   - Eliminar la producción $A \to B$.

   Ahora bien, si G es una gramática $\lambda-$libre, y si S es el estado inicial, S no puede estar en la derecha de ninguna producción. En particular, no existen producciones simples de la forma $A \to S$. Por tanto, toda producción simple $A \to B$ de $\mathcal{P}$ verifica que $B \neq S$. Por otro lado, como G es $\lambda-$ libre, para toda producción $B \neq S$, las producciones de la forma $B \to x$ verifican que $x \neq \lambda$. Por tanto, ninguna de las producciones que añadimos en este proceso es una $\lambda-$producción. Y, por tanto, G_2 sigue siendo $\lambda-$libre.

c. Basta con observar que el algoritmo de eliminación de símbolos inútiles solamente elimina símbolos y producciones que los involucran. Pero no añade producciones. Por ello, si no había en G ninguna producción simple, tampoco la habrá en G_2. Si G era $\lambda-$libre, también lo será G_2 puesto que no añadimos $\lambda-$producciones.

Estas observaciones terminan la demostración del lema. ∎

Despues de todo nuestro trabajo, podemos finalmente demostrar un gran teorema que garantiza que toda gramática libre de contexto es equivalente a una gramática propia.

Teorema 93 *Toda gramática libre de contexto es equivalente a una gramática propia. Dicha equivalencia es calculable algorítmicamente.*

Algorithm 10 Algoritmo para encontrar una gramatica equivalente propia

Input: Una gramática $G := (V, \Sigma, S, \mathcal{P})$

Output: Una gramatica G_3 propia

Hallar G_1 la gramática $\lambda-$libre obtenida aplicando a G el algoritmo del Teorema 86.

Hallar la gramática G_2 al aplicar a G_1 el algoritmo del Teorema 89.

Hallar la gramática G_3 de aplicar a G_2 el algoritmo del Teorema 84.

Demostración. Basta con unir los algoritmos antes expuestos en el orden adecuado que indica el Lema 92 anterior. El proceso será el siguiente:
El algoritmo anterior realiza la tarea prescrita. La gramática G_1 es claramente $\lambda-$libre como consecuencia del Teorema 86. Como consecuencia del apartado 2 del Lema 92, como la gramática G_1 es $\lambda-$libre, también es $\lambda-$libre la gramática G_2. Por otro lado, el Teorema 89 nos garantiza que G_2 es libre de producciones simples. Como consecuencia del apartado 3 del Lema 92, la gramática G_3 sigue siendo una gramática $\lambda-$libre y libre de producciones simples. Asimismo, el Teorema 84 nos garantiza que G_3 es libre de símbolos inútiles.
Finalmente, el apartado 1 del Lema 92, nos garantiza que G_3 es acíclica. Por tanto, verifica todas las propiedades para ser una gramática propia. ■

4.5. El Problema de Palabra

En esta subsección mostaremos que el problema de saber si una palabra en concreto es generada por una gramática libre de contexto es decidible para gramáticas libres de contexto. Es decir, se trata de mostrar un algoritmo que resuelve el problema siguiente:

Problema de palabra para gramáticas libres de contexto. Dada una gramática libre de contexto $G := (V, \Sigma, S, \mathcal{P})$ y dada una palabra $x \in \Sigma^*$, decidir si $x \in L(G)$.

Lema 94 *Sea $G := (V, \Sigma, S, \mathcal{P})$ una gramática libre de contexto y $\lambda-$libre. Sea $y \in L(G)$ una palabra aceptada por la gramática y sea:*

$$S \to x_1 \to x_2 \to \cdots \to x_n = y,$$

una derivación de y, donde $x_i \in (V \cup \Sigma)^*$ son formas sentenciales de la gramática. Entonces, la longitud de cada una de estas formas sentenciales verifica:

$$|x_i| \leq |y|, \ \forall i.$$

Demostración. De hecho, basta con observar que si tenemos dos formas sentenciales $x_i \to x_{i+1}$ y si la gramática es $\lambda-$libre, entonces o bien $x'_{i+1} = \lambda$ (en cuyo caso $x_i = S$ forzosamente) o bien $|x_i| \leq |x_{i+1}|$ (dado que no suprimimos ninguna variable,

al reemplazarla nos sale, al menos, un símbolo y la longitud no puede disminuir). Esta observación termina la demostración. ■

Gracias a este pequeño resultado, vamos a demostrar el teorema importante, esto es, que el problema de la palabra es resoluble por un algoritmo.

Teorema 95 *El problema de palabra es decidible para gramáticas libres de contexto.*

Demostración. Basta con usar el lema anterior. El procedimiento es el siguiente: en primer lugar, transformamos nuestra gramática original en una gramática $\lambda-$libre. posteriormente, dado x, construiremos un grafo $G_x := (N_x, E_x)$ usando las reglas siguientes:

- Los vértices del grafo N_x son todas las palabras de $(V \cup \Sigma)^*$ de longitud menor o igual que la longitud de x.
- Las aristas del grafo E_x son los pares (x, y) tales que $x, y \in N_x$ y $x \to^* y$.

Remarquemos que el grafo que estamos construyendo tiene como nodos las formas sentenciales de una gramática y dos elementos son vecinos si se puede rescribir uno a partir del otro. A partir de del grafo G_x, calculamos la componente conexa que contiene a S. Entonces, usando el lema anterior, x está en $L(G)$ si y solamente si hay un camino en el grafo G_x del nodo S a x. Calcular estos caminos es trivial, ya que el grafo tiene un número finito de nodos[2] y por lo tanto terminamos la demostración ■

Nota 96 *Decidibilidad no significa eficiencia. Es decir, el hecho de la existencia de un algoritmo para el problema de palabra no significa de modo inmediato que se pueda usar ese algoritmo para la detección de errores. De hecho, debe haber un pre–procesamiento para el cálculo del autómata (con pila, como veremos) que decide el problema de palabra y luego sólo debe usarse el autómata para cada palabra concreta. En otro caso estaríamos consumiendo una enormidad de tiempo de cálculo para cada verificación de correctitud de una palabra.*

Seguramente el lector se estará preguntando si no podemos mejorar la eficiencia del algoritmo para algún tipo especial de gramáticas libres de contexto. Por ejemplo, el caso de gramáticas lineas es muy sencillo, ¿algo parecido no puede existir para gramáticas libres de contexto?
Afortunadamente existe algo "parecido". La idea surge como una aplicación de programación dinámica y las gramáticas que se estudian son las llamadas "gramáticas en forma normal de Chomsky".

Definición 97 (Forma Normal de Chomsky) *Una gramática libre de contexto $G := (V, \Sigma, S, \mathcal{P})$ se dice que está en forma normal de Chomsky si es $\lambda-$libre y las únicas producciones (exceptuando, eventualmente, la única $\lambda-$ producción $S \mapsto \lambda$), son exclusivamente de uno de los dos tipos siguientes.*

- *$A \mapsto a$, con $A \in V$ y $a \in \Sigma$,*

[2] dejamos al alumno que calcule el número de nodos que tiene este grafo.

- $A \mapsto BC$, *con* $A, B, C \in V$.

Nótese que es acíclica porque carece de producciones unarias. En la definición, bien podríamos haber supuesto que es propia sin cambiar la esencia de la definición. Hemos dejado la habitual.
De otro lado, debe señalarse que el modelo se corresponde al modelo de codificación de polinomios (en este caso sobre anillos no conmutativos) llamado *"straight–line program"*.

Teorema 98 (Transformación a forma normal de Chomsky) *Toda gramática libre de contexto es equivalente a una gramática libre de contexto en forma normal de Chomsky. Además, esta equivalencia es algorítmicamente computable.*

Demostración. Bastará con que demos un algoritmo que transforma gramáticas propias en gramáticas en forma normal de Chomsky. Así, supongamos que tenemos una gramática $G := (V, \Sigma, S, \mathcal{P})$ propia. Procederemos del modo siguiente:
Definamos un par de clases V' y $\mathcal{P}'$ de símbolos no terminales y producciones, conforme a las reglas siguientes:

- Inicializar con $V' := V$, $\mathcal{P}' := \emptyset$.
- Si $S \to \lambda$ está en $\mathcal{P}$, añadir $S \to \lambda$ a $\mathcal{P}'$.
- Si en $\mathcal{P}$ hay una producción del tipo $A \to a$ donde $a \in \Sigma$ entonces, añadir $A \to a$ a V'.
- Si en $\mathcal{P}$ hay una producción del tipo $A \to BC$, con $B, C \in V$, añadir la misma producción a $\mathcal{P}'$.
- Finalmente, para cada producción en $\mathcal{P}$ del tipo

  $$A \to x_1 \dots x_k$$

  con $x_i \in V \cup \Sigma$ que no sea de ninguno de los tres tipos anteriores[3] realizar las tareas siguientes:

  - Para cada i tal que $x_i \in V$, no modificar V'
  - Para cada i tal que $x_i \in \Sigma$, añadir a V' una nueva variable C_i, distinta a todas las que ya estuvieran en V'. Añadir a $\mathcal{P}'$ la producción $C_i \to x_i$ en este caso. [4]
  - Añadir a $\mathcal{P}'$ la producción $A \to y_1 \cdots y_k$, donde y_i viene dada por:

    $$y_i := \begin{cases} x_i, & \text{si } x_i \in V \\ C_i, & \text{en otro caso} \end{cases}$$

[3] Obsérvese que $k \geq 2$ puesto que no hay producciones simples. Si $k = 2$, al no ser de ninguno de los tipos anteriores, son o bien dos símbolos en Σ o bien uno es un símbolos en Σ y el otro está en V. En cualquier caso se aplica el mismo método.

[4] Observese que, en este caso, aparece una producción del Tipo 1

- Si $k = 2$, no modificar.
- Si $k > 2$, reemplazar la producción $A \to y_1 \cdots y_k$ por una cadena de producciones:

$$A \to y_1 C_1, C_1 \to y_2 C_2, \ldots, C_{k-1} \to y_{k-1} y_k$$

añadiendo a V' las variables $\{C_2, \ldots, C_{k-1}\}$.

- OUTPUT: $(V', \Sigma, S, \mathcal{P}')$.

Está claro que el algoritmo descrito verifica las propiedades deseadas. ■

Nota 99 *Obsérvese que los árboles de derivación asociados a gramáticas en forma normal de Chomsky son árboles binarios cuyas hojas vienen de nodos con salida unaria.*

El último comentario es muy interesante para el algoritmo que resuelve el problema de palabra para gramáticas en forma normal de Chomsky. Esto lo veremos en un capítulo próximo.

4.5.1. Forma Normal de Greibach

Es otra "normalización" de las gramáticas libres de contexto que hace referencia al trabajo de Sheila A. Greibach en Teoría de Autómatas. Si la Forma Normal de Chomsky se corresponde con la presentación de polinomios como straight–line programs, la forma de Greibach se corresponde a la codificación mediante monomios.

Definición 100 (Producciones Monomiales) *Una gramática $G := (V, \Sigma, S, \mathcal{P})$ se dice en forma normal de Greibach si es $\lambda-$libre y las únicas producciones (exceptuando, eventualmente, la única $\lambda-$ producción $S \mapsto \lambda$) pertenecen al tipo siguiente:*

$$A \mapsto ax,$$

donde $A \in V$ es una variable, $a \in \Sigma$ es un símbolo terminal (posiblemente λ) y $x \in V^*$ es una lista (posiblemente vacía) de símbolos no terminales entre los cuales no está el símbolo inicial S.

No es para nada obvio, pero se puede conseguir una forma normal de Greibach en tiempo polinomial.

4.6. Cuestiones y problemas

4.6.1. Cuestiones

Cuestión 21 *Comprobar, utilizando las siguientes producciones de una gramática G, que al convertir una gramática a $\lambda-$libre, puede quedar con símbolos inútiles:*

$$S \mapsto a \mid aA, A \mapsto bB, B \mapsto \lambda.$$

Cuestión 22 *Decidir si existe un algoritmo que realice la tarea siguiente:*
Dada una gramática libre de contexto G y dadas dos formas sentenciales de la gramática c y c', el algoritmo decide si $c \vdash_G c'$.

Cuestión 23 *Sean L_1 y L_2 dos lenguajes libres de contexto. Decidir si es libre de contexto el lenguaje siguiente:*

$$L := \bigcup_{n\geq 1} (L_1)^n (L_2)^n.$$

Cuestión 24 *Hallar una estimación del número de pasos necesarios para generar una palabra de un lenguaje libre de contexto, en el caso en que la gramática que lo genera esté en forma normal de Chomsky.*

Cuestión 25 *Discutir si alguno de los siguientes lenguajes es un lenguaje libre de contexto:*

a. $\{\omega \in \{a,b\}^* \ : \ \omega = \omega^R, \ \forall x, y \in \{a,b\}^*, \omega \neq xabay\}$.

b. $\{a^i b^j c^k \ : \ i = j \vee j = k\}$.

c. $\{a^i b^j c^k d^l \ : \ (i = j \wedge k = l) \vee (i = l \wedge j = k)\}$.

d. $\{xcy \ : \ x, y \in \{a,b\}^* \wedge \sharp_a(x) + \sharp_b(y) \in 2\mathbb{Z} \wedge |x| = |y|\}$.

4.6.2. Problemas

Problema 50 *Dada una gramática libre de contexto G, con las siguientes producciones:*

$$S \mapsto AB \mid 0S1 \mid A \mid C, A \mapsto 0AB \mid \lambda, B \mapsto B1 \mid \lambda.$$

Se pide:

- *Eliminar los símbolos inútiles.*
- *Convertirla en $\lambda-$libre.*
- *Eliminar las producciones simples.*

Problema 51 *Eliminar las variables improductivas en la gramática G con las siguientes producciones:*

$$S \mapsto A \mid AA \mid AAA, \ A \mapsto ABa \mid ACa \mid a, B \mapsto ABa \mid Ab \mid \lambda,$$

$$C \mapsto Cab \mid CC, D \mapsto CD \mid Cd \mid CEa, E \mapsto b.$$

Eliminar los símbolos inaccesibles en la gramática resultante.

Problema 52 *Hallar una gramática $\lambda-$libre equivalente a la siguiente:*

$$S \mapsto aSa \mid bSb \mid aAb \mid bAa, A \mapsto aA \mid bA \mid \lambda.$$

¿Es una gramática propia?

Problema 53 *Hallar una gramática propia equivalente a la siguiente:*

$$S \mapsto XY, X \mapsto aXb \mid \lambda, Y \mapsto bYc \mid \lambda.$$

Problema 54 *Sea $G = (V, \Sigma, S, P)$ la gramática libre de contexto dada por las propiedades siguientes:*

- $V := \{S, X, Y, Z, T\}$,
- $\Sigma := \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, \times, (,)\}$,
- *Las producciones en P están dadas por:*

$$S \mapsto X \mid X + S, X \mapsto T \mid Y \times Z,$$

$$Y \mapsto T \mid (X + S), Z \mapsto Y \mid Y \times Z,$$

$$T \mapsto 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9.$$

Se pide:

a. *Hallar la clase de formas terminales de esta gramática.*

b. *Hallar el lenguaje generado por esta gramática.*

c. *Eliminar producciones unarias.*

d. *Eliminar producciones inútiles.*

e. *Convertirla en una gramática propia.*

f. *Transformarla en forma Normal de Chomsky.*

Problema 55 *Hacer los mismos pasos 3 a 6 del problema anterior con la gramática siguiente:*

$$S \mapsto A \mid B, A \mapsto aA \mid aAb \mid a, B \mapsto Bd \mid ABa \mid b.$$

Problema 56 *Eliminar $\lambda-$producciones y hacer los mismos pasos del problema anterior con la gramática siguiente:*

$$S \mapsto ABS \mid BAS \mid \lambda, A \mapsto bAA \mid a, B \mapsto aBB \mid b.$$

Problema 57 *Dar un algoritmo que decida si el lenguaje generado por una gramática libre de contexto es finito o infinito.*

Capítulo 5

Autómatas con Pila y Aplicaciones

> Nick: [throws a stack of paper on his desk into the garbage]
> Connor: Oh... actually, that's my dissertation. Yeah.
> Nick: [retrieves the stack of papers from the garbage and begins paging through it]
> Connor: See, I argued that all life on Earth derived from organisms carried here by alien spacecraft. It's pretty sexy stuff.
> Nick: [throws stack of papers back in the garbage]
> Connor: ...it's a work in progress, really...
>
> De la serie "Primeval"

La cita de hoy viene de una serie de ciencia ficción británica, donde se nos presentan a dos personajes. El primero tira todos los papeles que aparecen en la mesa, pensando que es basura. Mientras Connor (el segundo personaje) le corrige y le dice que es su tesis doctoral. Nick recupera los folios cuando oye el tema de la tesis doctoral y decide tirar la tesis doctoral de su compañero a la papelera. La papelera en esta ocasión hace las funciones de pila donde se pueden echar cosas de una manera efectiva. También se puede recuperar un objeto pero eso requiere sacar todas las cosas que están encima del objeto que se quiere recuperar. Y en este caso (y en todos los casos), hay dos funciones básicas que realiza la pila, que son apilar y desapilar. Veremos en la siguiente sección como se representan estas operaciones en un lenguaje formal y como se relacionan con la teoría de autómatas.

5.1. Nociones Básicas

Antes de pasar a definir un autómata con pila (o *Pushdown Automata*, PDA) recordemos (superficialmente) la estructura de datos pila (stack) vista como lenguaje formal y las funciones y relaciones que la caracterizan.
Podemos identificar las pilas con ciertos lenguajes formales sobre un nuevo alfabeto Γ. Comenzaremos añadiendo un nuevo símbolo Z_0 que hará las funciones de marcar el fondo de la pila. Las pilas (stacks) son elementos del lenguaje: $Z_0 \cdot \Gamma^*$. El símbolo Z_0 se identificará con el significado *Fondo de la Pila.*[1]
Tendremos unas ciertas funciones sobre pilas:

- empty : Definimos la aplicación

  $$\text{empty} : Z_0 \cdot \Gamma^* \longrightarrow \{0, 1\},$$

  dada mediante:

  $$\text{empty}(Z_0 w) = \begin{cases} 1 & w = \lambda, \\ 0 & w \neq \lambda. \end{cases}$$

- top : Definimos la aplicación

  $$\text{top} : Z_0 \cdot \Gamma^* \longrightarrow \Gamma \cup \{\lambda\},$$

  dada mediante:

  $$\text{top}(Z_0 w) = \begin{cases} Z_0 & w = \lambda, \\ c & w = yc. \end{cases}$$

  Obsérvese que hemos elegido leer Z_0 cuando la pila está vacía[2]

- push : Apilar (empujar) una pila encima de otra. Definimos la aplicación

  $$\text{push} : Z_0 \cdot \Gamma^* \times \Gamma^* \longrightarrow Z_0 \cdot \Gamma^*,$$

  mediante la regla siguiente:
  Dada una pila $Z_0 \cdot w \in Z_0 \cdot \Gamma^*$, y una palabra $y \in \Gamma^*$, definimos

  $$\text{push}(Z_0 \cdot w, y) := Z_0 \cdot wy \in Z_0 \cdot \Gamma^*.$$

- pop *(Pull Out the toP):* Definimos la aplicación

  $$\text{pop} : Z_0 \cdot \Gamma^* \longrightarrow Z_0 \cdot \Gamma^*,$$

  mediante la regla siguiente:
  Dada una pila $Z_0 w \in Z_0 \cdot \Gamma^*$, definimos $\text{pop}(Z_0 w)$ como el resultado de eliminar $\text{top}(Z_0 w)$, esto es

[1] El símbolo de 'fondo de la pila' no está suficientemente estandarizado. Diversos autores usan diveras variantes de símbolos como $\sharp, \$, []$ y otros. Usaremos Z_0 como la simplificación menos molesta de todas esas variaciones.

[2] Podríamos también haber definido $\text{top}(Z_0) = \lambda$. Esta corrección subjetiva la hemos hecho para enfatizar el hecho de que la pila está vacía. En caso de hacer esta elección, deberían modificarse todas las notaciones que siguen de modo conforme.

$$\mathrm{pop}(Z_0 w) = \begin{cases} Z_0 c_1 \cdots c_{n-1} & w = c_1 \cdots c_n, \\ Z_0 & w = \lambda. \end{cases}$$

Nota 101 *Una de las propiedades básicas de las operaciones es, obviamente, la siguiente:*

$$\mathrm{push}(\mathrm{pop}(Z_0 w), \mathrm{top}(Z_0 w)) = Z_0 w.$$

Se deja al lector la prueba de este simple ejercicio, aunque se dice también como pista que se resuelve facilmente por inducción.

Este capítulo trata sobre autómatas con pila, que son autómatas finitos más una memoria infinita donde se puede acceder solamente al primer elemento de ella. Una pregunta que podríamos hacernos ahora es porque poner una memoria donde solamente se pueda acceder al contenido de una manera tan restrictiva.
Anticiparemos la respuesta: los autómatas con pilas son los modelos básicos para resolver el problema de la palabra para lenguajes incontextuales. La definición formal de estos autómatas está dada en el siguiente bloque.

Definición 102 (Non–Deterministic Pushdown Automata) *Un autómata con pila (o Pushdown Automata) indeterminista es una lista* $A := (Q, \Sigma, \Gamma, q_0, F, Z_0, \delta)$ *donde:*

- Q *es un conjunto finito cuyos elementos se llaman estados y que suele denominarse espacio de estados,*
- Σ *es un conjunto finito (alfabeto),*
- Γ *es un conjunto finito llamado "alfabeto de la pila",*
- q_0 *es un elemento de* Q *que se denomina estado inicial,*
- F *es un subconjunto de* Q*, cuyos elementos se denominan estados finales aceptadores,*
- Z_0 *es un símbolo que está en el alfabeto* Γ *y que se denomina "fondo de la pila",*
- δ *es una correspondencia:*

  $$Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \longrightarrow Q \times \Gamma^*,$$

  que se denomina función de transición y que ha de verificar la propiedad siguiente[3] *para cada lista* $(q, a, c) \in Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$*:*

  $$\sharp\left(\{(q', w) \in Q \times \Gamma^* \mid (q', w) \in \delta(q, a, c)\}\right) < \infty.$$

  Es decir, sólo un número finito de elementos de $Q \times \Gamma^*$ *estarán relacionados con cada elemento* (q, a, c) *mediante la función de transición.*

[3] Recuérdese que si X es un conjunto finito, denotamos por $\sharp(X)$ su cardinal (i.e. el número de sus elementos).

Nota 103 *De nuevo, como en el caso de autómatas finitos indeterministas, hemos preferido usar una notación funcional menos correcta del tipo*

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{Z_0\}) \longrightarrow Q \times \Gamma^*,$$

para representar correspondencias, que la notación como aplicación más correcta:

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{Z_0\}) \longrightarrow \mathcal{P}(Q \times \Gamma^*).$$

La notación elegida pretende, sobre todo, enfatizar la diferencia entre el caso determinístico (δ es aplicación) frente al indeterminístico (δ es correspondencia).

Nota 104 *Nótese que hemos supuesto que la función de transición δ tiene su rango (el conjunto hacia el que va a parar) en $Q \times \Gamma^*$. Esta condición nos dirá (más adelante) que no podemos escribir en la pila el símbolo de "fondo de pila" nada más que cuando se escriba en la configuración inicial. Podremos, sin embargo, leerlo. No estará, en ningún caso, "en medio" de la pila.*

El determinismo en autómatas con pila difiere del caso de autómatas finitos. No vamos a exigir que δ sea aplicación sino algo más delicado.

Definición 105 (Autómata con pila determinista) *Un autómata con pila $A := (Q, \Sigma, \Gamma, q_0, F, Z_0, \delta)$ se denomina determinista si verifica las siguientes dos propiedades:*

- *La imagen de cualquier lectura contiene a lo sumo 1 elemento.*
- *Si dados $q \in Q$ y $c \in \Gamma$, existieran $(q, w) \in Q \times \Gamma^*$ tales que $\delta(q, \lambda, c) = (q', w)$, entonces, ninguno de los elementos de $Q \times \Sigma \times (\Gamma \cup \{Z_0\})$ tiene imagen por δ.*

Notar que estas condiciones son necesarias para definir determinismo. En los autómatas finitos, era posible eliminar $\lambda-$transiciones ya que no habia memoria de la que leer. En este caso, debemos estar atentos a las transiciones donde solo leemos la memoria. También queremos remarcar que, de la misma forma que se podia definir el lenguaje aceptado por un autómata finito, también podemos definir el lenguaje aceptado por un autómata con pila.
Hay dos maneras de interpretar el lenguaje aceptado por un autómata con pila: por estado final aceptador y por pila y cinta vacías.
Veremos además que ambas nociones de lenguajes son equivalentes, aunque, posiblemente, con diferentes autómatas. Esto es, un lenguaje es aceptado por un autómata por estado final aceptador si y solamente si es aceptado por un autómata por cinta y pila vacías.
La razón de usar ambas nociones se justifica por la orientación de cada una. Así, el lenguaje aceptado por estado final aceptador extiende la noción de autómata finito y es extensible a nociones más abstractas de máquinas como los autómatas bidireccionales o las máquinas de Turing. Por su parte, ver los lenguajes como lenguajes aceptados por cinta y pila vacías simplifican probar que los lenguajes generados por gramáticas libres de contexto son aquellos aceptados por autómatas con pila.

Definición 106 *Sea $A := (Q, \Sigma, \Gamma, q_0, F, Z_0, \delta)$ un autómata con pila,*

- *$Q \times \Sigma^* \times \Gamma^*$ es el conjunto de descripciones instantaneas,*
- *La transición $\Rightarrow$ se define por las reglas siguientes:*

$$(q, x, w) \Rightarrow (q', y, z) \Leftrightarrow \exists a \in \Sigma \cup \{\lambda\} \text{ y } b \in \Gamma, \\ x = ay,\ w = bw'\ (q', z') = \delta(q, a, b),\ z = z'w'.$$

- *Denotaremos $(q, x, w) \Rightarrow^* (q', y, z)$ si exite una cadena de estados tales que,*

$$(q, x, w) \Rightarrow \ldots \Rightarrow (q', y, z).$$

- *Una descripción instantanea (q, λ, w) se dice final aceptadora si $q \in F$.*

Definición 107 (Lenguaje aceptado mediante estado final aceptador) *Sea $A := (Q, \Sigma, \Gamma, q_0, F, Z_0, \delta)$ un autómata con pila. Para cada palabra $x \in \Sigma^*$, definimos la descripcion instantanea inicial en x a la descripción instantanea:*

$$I_A(x) := (q_0, x, Z_0).$$

Llamaremos lenguaje aceptado (mediante estado final final aceptador) por el autómata A (y lo denotaremos por $L_f(A)$) al conjunto siguiente:

$$L_f(A) := \{x \in \Sigma^* \mid I_A(x) \Rightarrow^* (f, \lambda, w),\ f \in F\}.$$

Definición 108 (Lenguaje aceptado mediante pila y cinta vacías) *Sea $A := (Q, \Sigma, \Gamma, q_0, F, Z_0, \delta)$ un autómata con pila. Llamaremos lenguaje aceptado (mediante pila y cinta vacías) por el autómata A (y lo denotaremos por $L_\emptyset(A)$) al conjunto siguiente:*

$$L_\emptyset(A) := \{x \in \Sigma^* \ :\ I_A(x) \Rightarrow^* (f, \lambda, \lambda),\ f \in Q\}.$$

La diferencia entre aceptar por pila vacía o por estado final es que en el caso de aceptar por estado final, la pila puede estar o no vacía mientras que en el caso de aceptación por pila vacía, la pila queda vacía, pero no nos preocupamos de cuál es el estado alcanzado.
El siguiente resultado muestra la equivalencia entre ambas formas de aceptación, esto es, que un lenguaje que es aceptado por un autómata con pila por estado final aceptador es también aceptado por pila y cintas vacías por **otro** automata con pila y viceversa. Además es facilmente calculable un autómata a partir de otro.
Por lo tanto, no nos tendremos que preocupar si hay diferencias entre los lenguajes aceptados de una forma o de la otra y la elección será unicamente por gusto. En estos apuntes utilizaremos la segunda forma, ya que será más cómodo utilizar el caso de pila vacía al no tener que definir el conjunto de estados finales.

Proposición 109 *Un lenguaje es aceptado por algún autómata con pila mediante pila y cinta vacías si y solamente si es aceptado por algún autómata con pila (posiblemente otro distinto) mediante estado final aceptador. Es decir, Sea $A := (Q, \Sigma, \Gamma, q_0, F, Z_0, \delta)$ y sean $L_1 := L_f(A) \subseteq \Sigma^*$ y $L_2 := L_\emptyset(A) \subseteq \Sigma^*$, respectivamente los lenguajes aceptados por A mediante cinta y pila vacías o mediante estado final aceptador. Entonces, se tiene:*

a. *Existe un autómata con pila B tal que $L_1 = L_\emptyset(B)$,*

b. *Existe un autómata con pila C tal que $L_2 = L_f(C)$.*

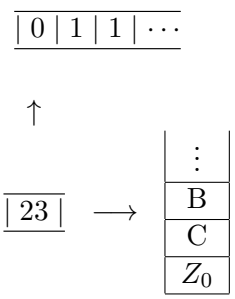
Dejaremos el lema sin demostración, por su complicación técnica.

Nota 110 *No es cierto, en el caso de autómatas con pila, que todo autómata con pila indeterminista sea equivalente a un autómata con pila determinista. Así, el siguiente lenguaje es aceptado por un autómata con pila indeterminista, pero no puede ser aceptado por un atómata con pila determinista:*

$$L := \{a^n b^m c^n \ : \ n, m \geq 1\} \bigcup \{a^n b^m c^m \ : \ n, m \geq 1\} \subseteq \{a, b, c\}^*.$$

La idea es la siguiente: despues de leer todas las letras $a^n b^n$, no podemos decidir cuantas c van a aparecer. Esta es una de las caracteristicas principales de los lenguajes aceptados por un automata con pila.

Tanto los contenidos de la cinta, la pila y el estado dan una representación del autómata con pila de una forma que generaliza la representación de un autómata finito. En este ejemplo, el alfabeto de la "cinta" es $\Sigma = \{0, 1\}$. El alfabeto de la pila será $\Gamma := \{A, B, C, Z_0\}$. El espacio de estados serán los primeros cien números naturales, i.e. $Q = \{0, 1, 2, 3, \ldots, 98, 99, 100\}$. Una representación gráfica de la configuración será:



En este dibujo, el estado es 23 y la unidad de control "lee" el primer símbolo de la cinta de entrada 0 y también "lee" el símbolo B en la pila. Supondremos que, en la pila, "lee" siempre el símbolo que se encuentra más arriba en la pila, lo que supone que, en el dibujo, por encima de B no hay nada en la pila.
Despúes de este pequeño inciso, volvamos al modelo general y clasifiquemos los diferentes tipos de transiciones. La clasificación dependerá de que operaciones realicemos en la pila y en la cinta.

- **Transiciones Read/Push.** Son transiciones entre dos configuraciones:

$$(q, x, Z_0 w) \Rightarrow^* (q', y, Z_0 z),$$

donde $q, q' \in Q$ y $x := a_1 \cdots a_n$. Realizamos las siguientes operaciones:

- Read Leemos la información $(q_0, a_1, \text{top}(Z_0 w))$. Supondremos $a_1 \neq \lambda$ y trataremos el caso opuesto por separado.

- Transition Aplicamos la función de transición $\delta(q_0, a_1, \text{top}(Z_0 w)) = (q', w')$, con $w' \in \Gamma^*$, $w' \neq \lambda$.
- Push and Move Entonces,
  - cambiamos el estado de la configuración a q',
  - $z := a_2 \cdots a_n$ ("borramos" un símbolo de la palabra a analizar (move)).
  - $Z_0 z := \text{push}(\text{pop}(Z_0 w), w')$.

- **Transiciones Read/Pop:** Son transiciones entre dos configuraciones:

$$(q, x, Z_0 w) \Rightarrow^* (q', y, Z_0 z),$$

donde $q, q' \in Q$ y $x := a_1 \cdots a_n$. Realizamos las siguientes operaciones:

- Read Leemos la información $(q_0, a_1, \text{top}(Z_0 w))$. Supondremos $a_1 \neq \lambda$ ya que este caso tiene un tratamiento especial.
- Transition Aplicamos la función de transición $\delta(q_0, a_1, \text{top}(Z_0 w)) = (q', \lambda)$. Esto quiere decir que debemos desapilar la cima de la pila.
- Pop and Move Entonces,
  - cambiamos el estado de la configuración a q',
  - $z := a_2 \cdots a_n$ ("borramos" un símbolo de la palabra a analizar (move)).
  - $Z_0 z := \text{pop}(Z_0 w)$.

- **Transiciones Lambda/Push.** Son transiciones entre dos configuraciones:

$$(q, x, Z_0 w) \Rightarrow^* (q', x, Z_0 z),$$

donde $q \in Q$ y $x := a_1 \cdots a_n$. Realizamos las siguientes operaciones:

- Read Lambda En este caso, no se lee la cinta aunque sí se lee la pila. Leemos $(q, \lambda, \text{top}(Z_0 w))$ (es decir, el estado, la palabra vacía y la cima de la pila).
- Transition Aplicamos la función de transición obteniendo (q', w'), con $w' \in \Gamma^*$, $w' \neq \lambda$.
- Push Entonces,
  - $q' := q$ (cambia el estado de la transición).
  - $Z_0 z := \text{push}(\text{pop}(Z_0 w), w')$.

- **Transiciones Lambda/Pop:** Son transiciones entre dos configuraciones:

$$(q, x, Z_0 w) \Rightarrow^* (q', x, Z_0 z),$$

donde $q \in Q$ y $x := a_1 \cdots a_n$. Realizamos las siguientes operaciones:

- Read Lambda De nuevo, no se lee la cinta aunque sí se lee la pila. Dejamos esta definición para el lector.

Nota 111 *Los autómatas finitos se pueden releer como autómatas con pila del modo siguiente: Suponemos que la función de transición δ verifica que*

$$\delta(q, a, c) = (q', c), \forall (q, a, c) \in Q \times (\Sigma \cup \{\lambda\}) \times \Gamma^*.$$

En este caso, todas las instrucciones pasan por apilar el elemento que se ve en la cima de la pila, lo que no cambia el contenido de la pila desde la configuración inicial.

Proposición 112 *Si A es un autómata con pila determinista, entonces dada una descripción instantanea s_1, existirá a lo sumo otra descripción instantanea s_2 tal que $s_1 \Rightarrow s_2$.*

La demostración se hace siguiendo estudiando por separado las dos hipótesis del autómata determinista. Se deja como ejercicio para el lector.

Nota 113 *Esta propiedad nos garantiza que la ejecución de un autómata con pila determinista es posible dado que a cada configuración le sigue o bien una única configuración siguiente o bien una salida (hacia estado final aceptador o de rechazo) por no tener ninguna opción de continuar.*

La introducción que hemos hecho de la noción de Autómata con Pila se basa en varios principios básicos. El primero es que los Autómatas con Pila son "expresables" sobre un alfabeto finito, del mismo modo que lo fueron los autómatas. En este punto, el lector tiene que tener claro los ingredientes de un autómata con pila y entender la forma de computar que se asocia a un autómata con pila. El lector no tendrá reparos para admitir que se trata de un programa que es ejecutable en algún tipo de intérprete que sea capaz de seguir las "instrucciones" asociadas a la transición. Un buen ejercicio es diseñar un programa (en Java, C++, C o cualquier lenguaje) asociado a un autómata (con la salvedad de los problemas de determinismo/indeterminismo).
Sin embargo, podemos utilizar diversas representaciones del autómata. La más simple es la de tabla. Para ello, podemos usar dos entradas. De una parte, el producto cartesiano $E := Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{Z_0\})$ que será el conjunto de las entradas de la función de transición. De otro lado tenemos un conjunto infinito de las "salidas" $O := Q \times \Gamma^*$, pero nuestra hipótesis nos dice que sólo un número finito de elementos de $Q \times \Gamma^*$ van a entrar en la relación. Así tendremos una tabla como la siguiente:

Ejemplo 14 *El PDA viene dado por:*

- $Q := \{q_0, q_1, r, s\}$,
- $\Sigma := \{0, 1\}$,
- $\Gamma := \{A, Z_0\}$,
- *Estado Inicial* q_0.
- *Símbolo de fondo de pila* Z_0.

- $F := \{r\}$.

Tabla de transición:

E	**O**
$(q_0, 0, Z_0)$	(q_0, A)
$(q_0, 0, A)$	(q_0, AA)
$(q_0, 1, A)$	(q_1, λ)
$(q_1, 1, A)$	(q_1, λ)
(q_1, λ, Z_0)	(r, λ)
(q_1, λ, A)	(s, λ)
$(q_1, 1, Z_0)$	(s, λ)

Tomemos como entrada la palabra $0^3 1^3 \in \Sigma^*$.
Inicializamos
$I := (q_0, 000111, Z_0)$
Ahora las sucesivas computaciones serán dadas por la secuencia siguiente:

- *Read/Push* $I \to c_1 = (q_0, 00111, Z_0 A)$
- *Read/Push* $c_1 \to c_2 = (q_0, 0111, Z_0 AA)$
- *Read/Push* $c_2 \to c_3 = (q_0, 111, Z_0 AAA)$
- *Read/Pop* $c_3 \to c_4 = (q_1, 11, Z_0 AA)$
- *Read/Pop* $c_4 \to c_5 = (q_1, 1, Z_0 A)$
- *Read/Pop* $c_5 \to c_6 = (q_1, \lambda, Z_0)$
- *Lambda/Pop* $c_6 \to c_6 = (r, \lambda, Z_0)$

Si, por el contrario, escojo la palabra $0^3 1^2 \in \Sigma^*$ se produce el efecto siguiente:
Inicializamos
$I := (q_0, 00011, Z_0)$

- *Read/Push* $I \to c_1 = (q_0, 0011, Z_0 A)$
- *Read/Push* $c_1 \to c_2 = (q_0, 011, Z_0 AA)$
- *Read/Push* $c_2 \to c_3 = (q_0, 11, Z_0 AAA)$
- *Read/Pop* $c_3 \to c_4 = (q_1, 1, Z_0 AA)$
- *Lambda/Pop* $c_4 \to c_5 = (q_1, \lambda, Z_0 A)$
- *Lambda/Pop* $c_5 \to c_6 = (s, \lambda, Z_0)$

Finalmente, escojo la palabra $0^2 1^3 \in \Sigma^*$ se produce el efecto siguiente:
Inicializamos
$I := (q_0, 00111, Z_0)$

- *Read/Push* $I \to c_1 = (q_0, 0111, Z_0A)$
- *Read/Push* $c_1 \to c_2 = (q_0, 111, Z_0AA)$
- *Read/Pop* $c_2 \to c_3 = (q_1, 11, Z_0A)$
- *Read/Pop* $c_3 \to c_4 = (q_1, 1, Z_0)$
- *Lambda/Pop* $c_4 \to c_5 = (s, \lambda, Z_0)$

Obsérvese que:

- Si tengo más ceros que unos, llegaré a leer (q_1, λ, A) con lo que acabaré en el estado s
- Si tengo más unos que ceros, llegaré a leer $(q_0, 1, z_0y)$ con lo que acabaré en s.
- La única forma de llegar a r sería tener el mismo número de ceros que de unos. Más aún, las únicas palabras que llegan al estado r son las dadas por

$$L := \{0^n 1^n \ : \ n \in \mathbb{N}\}.$$

■

Capítulo 6

Sucinta Introducción al Lenguaje de la Teoría Intuitiva de Conjuntos

Índice

6.1. Introducción

> Dios creo los números, el resto es cosa del hombre
>
> Anónimo

A finales del siglo XIX, G. Cantor introduce la Teoría de Conjuntos. Su propósito inicial es el modesto propósito de fundamentar matemáticamente el proceso de "contar". Eso sí, no se trataba solamente de contar conjuntos finitos sino también infinitos, observando, por ejemplo, que hay diversos infinitos posibles ($\aleph_0, 2^{\aleph_0}$ o $\aleph_1$, por ejemplo). Más allá del propósito inicial de Cantor, la Teoría de Conjuntos se transformó en un instrumento útil para las Matemáticas, como un lenguaje formal sobre el que escribir adecuadamente afirmaciones, razonamientos, definiciones, etc... Lo que aquí se pretende no es una introducción formal de la Teoría de Conjuntos. Para ello sería necesario hacer una presentación de los formalismos de Zermelo–Frænkel o de Gödel–Bernays, lo cual nos llevaría un tiempo excesivo y sería de todo punto infructuoso e ineficaz. Al contrario, pretendemos solamente unos rudimentos de lenguaje que nos serán de utilidad durante el curso, un "apaño", para poder

utilizar confortablemente el lenguaje si tener que produndizar en honduras innecesarias para la Ingeniería Informática. El recurso, también usado corrientemente en Matemáticas, es acudir a la Teoría Intuitiva de Conjuntos tal y como la concibe Haussdorff. Éste es el propósito de estas pocas páginas.

6.2. Los conjuntos y la pertenencia a conjuntos

> Conducir con orden mis pensamientos, empezando por los objetos más simples y más fáciles de conocer, para ascender poco a poco, gradualmente, hasta el conocimiento de los más complejos, y suponiendo incluso un orden entre ellos que no se parecen naturalmente unos a otros
>
> Rene Descartes

Comencemos considerando los conjuntos como conglomerados de objetos. Estos objetos pasarán a denominarse *elementos* y diremos que *pertenecen a un conjunto*. Para los objetos que no están en un conjunto dado, diremos que *no pertenecen al conjunto* (o no son elementos suyos).
Como regla general (aunque con excepciones, que se indicarán en cada caso), los conjuntos se denotan con letras mayúsculas:

$$\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{A}_1, \mathcal{A}_2, \ldots$$

mientras que los elementos se suelen denotar con letras minúsculas:

$$a, b, c, \ldots$$

El símbolo que denota pertenencia es $\in$ y escribiremos

$$a \in \mathcal{A}, \quad b \notin \mathcal{B},$$

para indicar "el elemento a pertenece al conjunto $\mathcal{A}$" y "el elemento b no pertenece al conjunto $\mathcal{B}$", respectivamente.
Hay muchas formas para definir un conjunto. Los símbolos que denotan conjunto son las dos llaves { y } y su descripción es lo que se escriba en medio de las llaves.

- *Por extensión:* La descripción de todos los elementos, uno tras otro, como, por ejemplo:
$$\mathcal{A} := \{0, 2, 4, 6, 8\}.$$
- *Por una Propiedad que se satisface:* Suele tomar la forma
$$\mathcal{A} := \{a \ : \ P(a)\},$$

donde P es una propiedad (una fórmula) que actúa sobre la variable a. Por ejemplo, el conjunto anterior puede describirse mediante:

$$\mathcal{A} := \{a \ : \ [a \in \mathbb{N}] \wedge [0 \leq a \leq 9] \wedge [2 \mid a]\},$$

donde hemos usado una serie de propiedades como $[a \in \mathbb{N}]$ (es un número natural), $[0 \leq a \leq 9]$ (entre 0 y 9), $[2 \mid s]$ (y es par). Todas ellas aparecen ligadas mediante la conectiva $\wedge$ (conjunción). Sobre la forma y requisitos de las propiedades no introduciremos grandes discusiones, no haremos restricciones sobre si es posible efectivamente calcular la pertenencia de un elemento.

Algunas observaciones preliminares.

- Existe un único conjunto que no tiene ningún elemento. Es el llamado **conjunto vacío** y lo denotaremos por $\emptyset$. La propiedad que verifica se expresa (usando cuantificadores) mediante:

$$\neg\left(\exists a, a \in \emptyset\right),$$

o también mediante la fórmula

$$\forall a, a \notin \emptyset.$$

- Aunque hemos dado un par de ejemplos de conjuntos, vemos que es difícil definir conjuntos si no contamos con otros conjuntos, ya que los elementos que están en nuestro conjuntos suelen ser tomados de otros conjuntos. Nuestro primer ejemplo tomaba elementos del conjunto de los números naturales. En realidad, a partir del conjunto vacío podemos definir los números naturales. La idea sería que el cero se corresponderia con el conjunto vacío, el uno se correspondería con el conjunto que contiene el conjunto vacio, el dos se correspondería con el conjunto que contiene el conjunto vacío y el conjunto que contiene el conjunto vacío y así sucesivamente.

- La **Estructura de Datos** relacionada con la noción de conjunto es el tipo `set`. Esta estructura de datos está diseñada para realizar de una manera eficiente las siguientes operaciones sobre conjuntos de elementos finitos:

  - determinar la pertenencia de un elemento a un conjunto,
  - hallar la unión de dos conjuntos,
  - hallar la intersección de dos conjuntos,
  - hallar el conjunto de los elementos que pertenecen a un conjunto pero no pertenecen a otro.

6.3. Operaciones elementales de conjuntos

> Los edificios se empiezan por los cimientos, no por el tejado
>
> Dicho popular

Hasta ahora solo hemos visto la definición del conjunto, con lo que no podemos llegar muy lejos. En esta sección veremos las operaciones y conceptos más comunes que se definen en conjuntos.
Se dice que un conjunto $\mathcal{A}$ está incluido (o contenido) en otro conjunto $\mathcal{B}$ si *todos los elementos de* $\mathcal{A}$ *son también elementos de* $\mathcal{B}$. También se dice que $\mathcal{A}$ es subconjunto de $\mathcal{B}$ en ese caso. Se usa el símbolo $\subseteq$ para indicar inclusión y la propiedad se "define" mediante:

$$\mathcal{A} \subseteq \mathcal{B} := [\forall a, a \in \mathcal{A} \Longrightarrow a \in \mathcal{B}] .$$

- Nótese la identificación entre la inclusión $\subseteq$ y la implicación $\Longrightarrow$ (o $\longrightarrow$, en la forma más convencional de la Lógica).
- Obviamente, a través de esa identificación, el conjunto vacío está contenido en cualquier conjunto. Es decir,
$$\emptyset \subseteq \mathcal{B},$$
para cualquier conjunto $\mathcal{B}$.
- Dos conjuntos se consideran iguales si poseen los mismos elementos. En términos formales:
$$(\mathcal{A} = \mathcal{B}) \iff ((\mathcal{A} \subseteq \mathcal{B}) \wedge (\mathcal{B} \subseteq \mathcal{A})) .$$
Lo que también puede escribirse con elementos mediante:
$$(\mathcal{A} = \mathcal{B}) \Leftrightarrow \forall a, ((a \in \mathcal{A}) \iff (a \in \mathcal{B})) .$$
- La familia de todos los subconjuntos de un conjunto $\mathcal{A}$ dado se denomina *las partes de* $\mathcal{A}$ y se suele denotar mediante $\mathcal{P}(\mathcal{A})$.

Ejemplo 15 *Es fácil, por ejemplo, mostrar la siguiente igualdad que describe las partes del conjunto* $\mathcal{A} := \{0, 1, 2\}$*:*

$$\mathcal{P}(\{0, 1, 2\}) = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\} .$$

No resulta tan fácil probar que la clase $\mathcal{P}(\mathbb{N})$ *es justamente el intervalo* $[0, 1]$ *de la recta real* $\mathbb{R}$*. Lo dejamos para más tarde (en forma puramente esquemática).*

Las conectivas lógicas del cálculo proposicional, permiten definir operaciones entre subconjuntos de un conjunto dado. Supongamos que tenemos un conjunto $\mathcal{A}$ dado y sean $\mathcal{B}, \mathcal{C} \in \mathcal{P}(\mathcal{A})$ dos de sus subconjuntos. Definimos:

- **Unión:**
$$\mathcal{B} \cup \mathcal{C} := \{a \in \mathcal{A} \ : \ (a \in \mathcal{B}) \vee (a \in \mathcal{C})\}.$$
- **Intersección:**
$$\mathcal{B} \cap \mathcal{C} := \{a \in \mathcal{A} \ : \ (a \in \mathcal{B}) \wedge (a \in \mathcal{C})\}.$$
- **Complementario:**
$$\mathcal{B}^c := \{a \in \mathcal{A} \ : \ a \notin \mathcal{B}\}.$$

Obsérvese que $\emptyset^c = \mathcal{A}$ y que $(\mathcal{B}^c)^c = \mathcal{B}$ para cualquier $\mathcal{B} \in \mathcal{P}(\mathcal{A})$.
Adicionalmente, podemos reencontrar la diferencia entre conjuntos y la traslación del EXCLUSIVE OR (denotado por XOR en Electrónica Digital) o por $\oplus$ (en Teoría de Números, hablando de restos enteros módulo 2, i.e. $\mathbb{Z}/2\mathbb{Z}$; aunque, en este caso se suele denotar simplemente mediante $+$).

- **Diferencia:**
$$\mathcal{B} \setminus \mathcal{C} := \{a \in \mathcal{A} \ : \ (a \in \mathcal{B}) \wedge (a \notin \mathcal{C})\}.$$

- **Diferencia Simétrica:**
$$\mathcal{B} \cup \mathcal{C} := \{a \in \mathcal{A} \ : \ (a \in \mathcal{B}) \oplus (a \in \mathcal{C})\}.$$

Las relaciones evidentes con estas definiciones se resumen en las siguientes fórmulas:

$$\mathcal{B} \setminus \mathcal{C} := \mathcal{B} \cap \mathcal{C}^c, \ \ \mathcal{B}\Delta\mathcal{C} = (\mathcal{B} \cup \mathcal{C}) \setminus (\mathcal{B} \cap \mathcal{C}) = (\mathcal{B} \setminus \mathcal{C}) \cup (\mathcal{C} \setminus \mathcal{B}).$$

Sería excesivo e innecesario expresar aquí con propiedad las nociones involucradas, pero dejemos constancia de la propiedades básicas de estas operaciones en las siguientes páginas.

Propiedades de la Unión.

Sean $\mathcal{B}, \mathcal{C}, \mathcal{D}$ subconjuntos de un conjunto $\mathcal{A}$, se tienen las siguientes propiedades:

- **Idempotencia:** $\mathcal{B} \cup \mathcal{B} = \mathcal{B}$, $\forall \mathcal{B} \in \mathcal{P}(\mathcal{A})$.
- **Asociativa:** $\mathcal{B} \cup (\mathcal{C} \cup \mathcal{D}) = (\mathcal{B} \cup \mathcal{C}) \cup \mathcal{D}$, $\forall \mathcal{B}, \mathcal{C}, \mathcal{D} \in \mathcal{P}(\mathcal{A})$.
- **Conmutativa:** $\mathcal{B} \cup \mathcal{C} = \mathcal{C} \cup \mathcal{B}$, $\forall \mathcal{B}, \mathcal{C} \in \mathcal{P}(\mathcal{A})$.
- **Existe Elemento Neutro:** El conjunto vacío $\emptyset$ es el elemento neutro para la unión:
$$\mathcal{B} \cup \emptyset = \emptyset \cup \mathcal{B} = \mathcal{B}, \quad \forall \mathcal{B} \in \mathcal{P}(\mathcal{A}).$$

La demostración de todas estas propiedades se realiza de una forma mecánica. Volvemos a dejar la demostración para el lector pero damos la idea en el caso de la idem potencia $\mathcal{B} \cup \mathcal{B} = \mathcal{B}$. La demostración se hace por doble contenido y normalmente hay un contenido que es trivial. En este caso,

$$\mathcal{B} \subseteq \mathcal{B} \cup \mathcal{B}.$$

Para demostrar que la parte izquierda esta contenida en la parte derecha, hay que demostrar que todo elemento $a \in \mathcal{B} \cup \mathcal{B}$ pertenece a $\mathcal{B}$. A partir de la definición, esto es trivial.
Veamos ahora las propiedades de la operación intersección.

Propiedades de la Intersección.

Sean $\mathcal{B}, \mathcal{C}, \mathcal{D}$ subconjuntos de un conjunto $\mathcal{A}$.

- **Idempotencia:** $\mathcal{B} \cap \mathcal{B} = \mathcal{B}$, $\forall \mathcal{B} \in \mathcal{P}(\mathcal{A})$.
- **Asociativa:** $\mathcal{B} \cap (\mathcal{C} \cap \mathcal{D}) = (\mathcal{B} \cap \mathcal{C}) \cap \mathcal{D}$, $\forall \mathcal{B}, \mathcal{C}, \mathcal{D} \in \mathcal{P}(\mathcal{A})$.
- **Conmutativa:** $\mathcal{B} \cap \mathcal{C} = \mathcal{C} \cap \mathcal{B}$, $\forall \mathcal{B}, \mathcal{C} \in \mathcal{P}(\mathcal{A})$.
- **Existe Elemento Neutro:** El conjunto vacío $\mathcal{A}$ es el elemento neutro para la unión:
$$\mathcal{B} \cap \mathcal{A} = \mathcal{A} \cap \mathcal{B} = \mathcal{B}, \quad \forall \mathcal{B} \in \mathcal{P}(\mathcal{A}).$$

Como la práctica hace al maestro, dejamos la demostración de estas propiedades al lector.

Propiedades Distributivas.

Sean $\mathcal{B}, \mathcal{C}, \mathcal{D}$ subconjuntos de un conjunto $\mathcal{A}$.

- $\mathcal{B} \cup (\mathcal{C} \cap \mathcal{D}) = (\mathcal{B} \cup \mathcal{C}) \cap (\mathcal{B} \cup \mathcal{D})$, $\forall \mathcal{B}, \mathcal{C}, \mathcal{D} \in \mathcal{P}(\mathcal{A})$.
- $\mathcal{B} \cap (\mathcal{C} \cup \mathcal{D}) = (\mathcal{B} \cap \mathcal{C}) \cup (\mathcal{B} \cap \mathcal{D})$, $\forall \mathcal{B}, \mathcal{C}, \mathcal{D} \in \mathcal{P}(\mathcal{A})$.
- $\mathcal{B} \cap (\mathcal{C} \Delta \mathcal{D}) = (\mathcal{B} \cap \mathcal{C}) \Delta (\mathcal{B} \cap \mathcal{D})$, $\forall \mathcal{B}, \mathcal{C}, \mathcal{D} \in \mathcal{P}(\mathcal{A})$.

Leyes de Morgan.

Por ser completos con los clásicos, dejemos constancia de las Leyes de Morgan. Sean $\mathcal{B}, \mathcal{C}$ subconjuntos de un conjunto $\mathcal{A}$.

$$(\mathcal{B} \cap \mathcal{C})^c = (\mathcal{B}^c \cup \mathcal{C}^c), \quad (\mathcal{B} \cup \mathcal{C})^c = (\mathcal{B}^c \cap \mathcal{C}^c).$$

Generalizaciones de Unión e Intersección.

Tras todas estas propiedades, dejemos las definiciones y generalizaciones de la unión e intersección en el caso de varios (o muchos) subconjuntos de un conjunto dado. Nótese la identificación entre $\cup$, $\vee$ y el cuantificador existencial $\exists$ (y, del mismo modo, la identificación entre $\cap$, $\wedge$ y el cuantificador universal $\forall$.

Un número finito de uniones e intersecciones.

Dados $\mathcal{A}_1, \ldots, \mathcal{A}_n$ unos subconjuntos de un conjunto $\mathcal{A}$. Definimos:

$$\bigcup_{i=1}^{n} \mathcal{A}_i := \mathcal{A}_1 \cup \mathcal{A}_2 \cup \cdots \cup \mathcal{A}_n = \{a \in \mathcal{A} \ : \ \exists i,\ 1 \leq i \leq n,\ a \in \mathcal{A}_i\}.$$

$$\bigcup_{i=1}^{n} \mathcal{A}_i := \mathcal{A}_1 \cap \mathcal{A}_2 \cap \cdots \cap \mathcal{A}_n = \{a \in \mathcal{A} \ : \ \forall i,\ 1 \leq i \leq n,\ a \in \mathcal{A}_i\}.$$

Unión e Intersección de familias cualesquiera de subconjuntos.

Supongamos $\{\mathcal{A}_i \ : \ i \in I\}$ es una familia de subconjuntos de un conjunto $\mathcal{A}$. Definimos:

$$\bigcup_{i\in I} \mathcal{A}_i := \{a \in \mathcal{A} \ : \ \exists i \in I, \ a \in \mathcal{A}_i\},$$
$$\bigcap_{i\in I} \mathcal{A}_i := \{a \in \mathcal{A} \ : \ \forall i \in I, \ a \in \mathcal{A}_i\}.$$

En ocasiones nos veremos obligados a acudir a uniones e intersecciones de un número finito o de un número infinito de conjuntos, y los conjuntos que utilizaremos seleccionados por algún criterio. Para referirnos más comodamente a estos conjuntos utilizaremos los subindices en vez de escribiendo cada uno de los conjuntos.

Conjuntos y Subconjuntos: Grafos No orientados.

Los grafos han sido también estudiados en cursos anteriores por lo que no nos detendremos en demasia. Recordad simplemente que estos objetos matemáticos ya fueron utilizados por Leonard Euler para estudiar el problema de los puentes de Königsberg. La razón de utilizarlos es que proporcionan formas muy convenientes de describir estructuras y a la vez es eficiente para realizar computaciones.

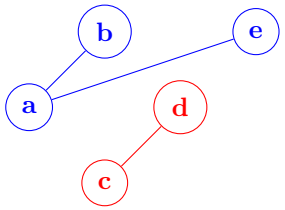
Definición 114 *Un grafo no orientado (o simplemente un grafo) es una lista* $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ *formada por dos objetos:*

- **Vértices:** *son los elementos del conjunto* $\mathcal{V}$ *(que usualmente se toma finito*[1]*) aunque podremos encontrar "grafos" con un conjunto "infinito" de vértices.*
- **Aristas:** *Es un conjunto de subconjuntos de* V*, es decir,* $\mathcal{E} \subseteq \mathcal{P}(\mathcal{V})$ *con la salvedad siguiente: los elementos* $\mathcal{B} \in \mathcal{E}$ *(que, recordemos, son subconjuntos de* $\mathcal{V}$*) que tienen dos elementos distintos.*

Ejemplo 16 *Un sencillo ejemplo sería:*

- **Vértices:** $\mathcal{V} := \{a, b, c, d, e\}$
- **Aristas:**
$$\mathcal{E} := \{\{a, b\}, \{a, e\}, \{c, d\}\} \subseteq \mathcal{P}(\mathcal{V}).$$

Al ser no orientado la matriz de adyacencia es simétrica y las componentes conexas son, también, subconjuntos de $\mathcal{V}$*, aunque de mayor cardinal. Gráficamente:*



[1] Aceptemos esta disgresión sin haber definido finitud

Nótese que no hemos definido la igualdad de grafos, ya que esto depende de la forma de llamar a los nodos. La idea es que dos grafos son iguales si se pueden dibujar de forma que sean iguales.
Tambien hacemos notar que podríamos haber aceptado aristas que van desde un nodo a sí mismo (tipo $\{a\}$ o $\{e\}$, por ejemplo) pero que el orden en que son descritos los elementos de una arista no es relevante: por eso hablamos de grafos no orientados. Las aristas de los grafos orientados son definidas mediante listas de dos nodos.

6.4. Producto cartesiano y conceptos derivados

> Y con este nuevo método, pude demostrar sin mucha dificultad muchos de los trabajosos teoremas que se conocen de la geometría clásica
>
> "Discurso del Método",Rene Descartes

Si en los grafos no orientados considerábamos aristas descritas de forma $\{a, b\}$ y el orden no interviene ($\{a, b\} = \{b, a\}$) ahora nos interesa destacar el papel jugado por el orden, hablamos de pares ordenados (a, b) y se corresponde al tipo de datos `list`. Así, por ejemplo, $(a, b) = (b, a)$ si y solamente si $a = b$. Una manera de representar las listas mediante conjuntos podría ser escribiendo (a, b) como abreviatura de $\{\{a\}, \{a, b\}\}$. Pero nos quedaremos con la intuición del tipo de datos `list`, donde cada dato es representado por varias celdas, donde cada celda contiene el elemento y un puntero a la siguiente celda.
Dados dos conjuntos $\mathcal{A}$ y $\mathcal{B}$ definimos el producto cartesiano de $\mathcal{A}$ y $\mathcal{B}$ como el conjunto de las listas de longitud 2 en las que el primer elemento está en el conjunto $\mathcal{A}$ y el segundo en $\mathcal{B}$. Formalmente,

$$\mathcal{A} \times \mathcal{B} := \{(a, b) \ : \ a \in \mathcal{A}, b \in \mathcal{B}\}.$$

Pero podemos considerar listas de mayor longitud: dados $\mathcal{A}_1, \ldots, \mathcal{A}_n$ definimos el producto cartesiano $\prod_{i=1}^{n} \mathcal{A}_i$ como las listas de longitud n, en las que la coordenada $i-$ésima está en el conjunto $\mathcal{A}_i$.

$$\prod_{i=1}^{n} \mathcal{A}_i := \{(a_1, \ldots, a_n) \ : \ a_i \in \mathcal{A}_i, \ 1 \leq i \leq n\}.$$

En ocasiones, se hacen productos cartesianos de familias no necesariamente finitas $\{\mathcal{A}_i \ : \ i \in I\}$ (como las sucesiones, con $I = \mathbb{N}$) y tenemos el conjunto:

$$\prod_{i=1}^{n} \mathcal{A}_i := \{(a_i : i \in I) \ : \ a_i \in \mathcal{A}_i, \ 1 \leq i \leq n\}.$$

En otras ocasiones se hace el producto cartesiano de un conjunto consigo mismo, mediante las siguientes reglas obvias:

$$\mathcal{A} = \mathcal{A}, \quad \mathcal{A}^2 := \mathcal{A} \times \mathcal{A}, \quad \mathcal{A}^i := \mathcal{A}^{i-1} \times i = \prod_{j=1}^{i} \mathcal{A}.$$

Algunos casos extremos de las potencias puedebn ser los siguientes:

- **Caso $i = 0$:** Para cualquier conjunto $\mathcal{A}$ se define $\mathcal{A}^0$ como el conjunto formado por un único elemento, que es el mismo independientemente de $\mathcal{A}$. Cuando hablemos de lenguajes, este elemento se conoce con la palabra vacía y se denota por λ. No se debe confundir $\mathcal{A}^0 := \{\lambda\}$ con el conjunto vacío $\emptyset$.
- **Caso $I = \mathbb{N}$:** Se trata de las sucesiones (infinitas numerables) cuyas coordenadas viven en $\mathcal{A}$. Se denota por $\mathcal{A}^{\mathbb{N}}$. Los alumnos han visto, en el caso $\mathcal{A} = \mathbb{R}$ el conjunto de todas las sucesiones de números reales (que se denota mediante $\mathcal{A}^{\mathbb{N}}$).

Aunque el concepto de producto cartesiano se aplica a todos los posibles conjuntos, en estos apuntes tendrá especial relevancia la siguiente aplicación.

Nota 115 (Palabras sobre un Alfabeto) *El conjunto de las palabras con alfabeto un conjunto $\mathcal{A}$ jugará un papel en este curso, se denota por $\mathcal{A}^*$ y se define mediante*

$$\mathcal{A}^* := \bigcup_{i \in \mathbb{N}} \mathcal{A}^i.$$

Volveremos con la noción más adelante.

Correspondencias.

Una correspondencia entre un conjunto $\mathcal{A}$ y otro conjunto $\mathcal{B}$ es un subconjunto $\mathcal{C}$ del producto cartesiano $\mathcal{A} \times \mathcal{B}$. En esencia es un grafo bipartito que hace interactuar los elementos de $\mathcal{A}$ con elementos de $\mathcal{B}$. Los elementos que interactúan entre sí son aquellos indicados por los pares que están en $\mathcal{C}$. En ocasiones se escribirán una notación funcional de la forma $\mathcal{C} : \mathcal{A} \longrightarrow \mathcal{B}$, aunque poniendo gran cuidado porque los subconjuntos de $\mathcal{C}$ no siempre definen funciones, ya que para cada elemento $x \in \mathcal{A}$ debe solamente existir un elemento $y \in \mathcal{B}$ tal que $(x, y) \in \mathcal{A} \times \mathcal{B}$.

Ejemplo 17 *Tomando $\mathcal{A} = \mathcal{B} = \mathbb{R}$, podemos definir la relación $R_1 \subseteq \mathbb{R}^2$ mediante:*

$$R_1 := \{(x, y) \in \mathbb{R}^2 \ : \ x = y^2\}.$$

Estaremos relacionando los número reales con sus raíces cuadradas. Obsérvese que los elementos x tales que $x < 0$ no están relacionados con ningún número y (no tienen raíz cuadrada real). El 0 se relaciona con un único número (su única raíz cuadrada) y los número reales positivos se relacionan con sus dos raices cuadradas.

En el siguiente ejemplo, veremos un subconjunto que si define una función. Con este ejemplo, tan parecido al anterior, queremos hacer notar la sutil diferencia que hay entre los dos casos.

Ejemplo 18 *Tomando los mismos conjuntos $\mathcal{A} = \mathcal{B} = \mathbb{R}$, podemos definir la relación $R_2 \subseteq \mathbb{R}^2$ distinta de la anterior:*

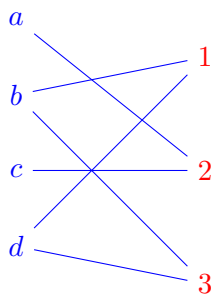
$$R_1 := \{(x, y) \in \mathbb{R}^2 \ : \ x^2 = y\}.$$

En este caso tenemos una función que relaciona cualquier x en $\mathbb{R}$ con su cuadrado.

Ejemplo 19 *Un grafo bipartito podría ser, por ejemplo, $\mathcal{A} := \{a, b, c, d\}$, $\mathcal{B} := \{1, 2, 3\}$ y una relación como $\mathcal{C} \subseteq \mathcal{A} \times \mathcal{B}$:*

$$\mathcal{C} := \{(a, 2), (b, 1), (b, 3), (c, 2), (d, 1), (d, 3)\},$$

cuyo grafo sería:



Nota 116 *En ocasiones abusaremos de la notación, escribiendo $\mathcal{C}(x) = y$ o $x\mathcal{C}y$, para indicar que los elementos $x \in \mathcal{A}$ e $y \in \mathcal{B}$ están en correspondencia a través de $\mathcal{C} \subseteq \mathcal{A} \times \mathcal{B}$.*

Relaciones.

Las relaciones son correspondencia $\mathcal{C} \subseteq \mathcal{A} \times \mathcal{A}$, es decir, aquellas correspondencias donde el conjunto de primeras coordenadas es el mismo que el conjunto de las segundas coordenadas.

Nota 117 (Una Relación no es sino un grafo orientado.) *Aunque, por hábito, se suele pensar en que los grafos orientados son relaciones sobre conjuntos finitos, pero admitiremos grafos con un conjunto infinito de vértices.*

Pongamos algunos ejemplos sencillos:

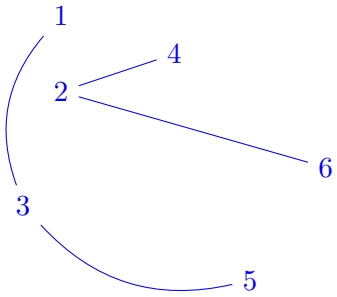
Ejemplo 20 (Un ejemplo al uso) *Consideremos el grafo $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ donde $\mathcal{V}$ es el conjunto de vértices dado por:*

$$\mathcal{V} := \{1, 2, 3, 4, 5, 6\},$$

y $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ es el conjunto de aristas orientadas siguiente:

$$\mathcal{E} := \{(1, 3), (3, 5), (2, 4), (2, 6)\}.$$

Gráficamente tendremos



Ejemplo 21 (La circunferencia unidad) *Es un grafo infinito cuyos vértices son los números reales* $\mathcal{V} = \mathbb{R}$ *y cuyas aristas son dadas mediante:*

$$\mathcal{E} := \{(x, y) \in \mathbb{R}^2 \ : \ x - y \in \mathbb{Z}\}.$$

No lo dibujaremos (tenemos demasiados vértices y demasiadas aristas) pero las componentes conexas están identicadas con los puntos de la circunferencia unidad

$$S^1 := \{(x, y) \in \mathbb{R}^2 \ : \ x^2 + y^2 - 1 = 0\}.$$

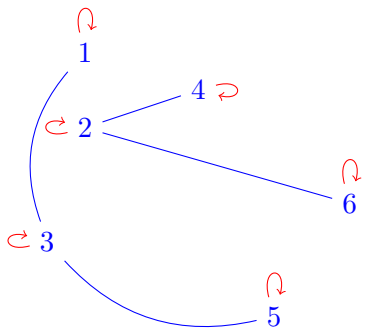
Algunos tipos de relaciones son más relevantes que otras por sus consecuencias. Destaquemos dos tipos especiales de relaciones: las relaciones de orden y de equivalencia.

Relaciones de Orden.

Son aquellas relaciones $\mathcal{C} \subseteq \mathcal{V} \times \mathcal{V}$, que verifican las propiedades siguientes:

- *Reflexiva:* $\forall x \in \mathcal{V}, \ \ (x, x) \in R$. La relación descrita en el Ejemplo 20 anterior no verifica esta propiedad. Para verificarla, se necesitaría que también fueran aristas las siguientes:

$$\{(1,1), (2,2), (3,3), (4,4), (5,5), (6,6)\} \subseteq \mathcal{E}.$$



  En cambio el ejemplo de la circunferencia verifica la propiedad reflexiva.

- *Antisimétrica:* Que se expresa mediante:

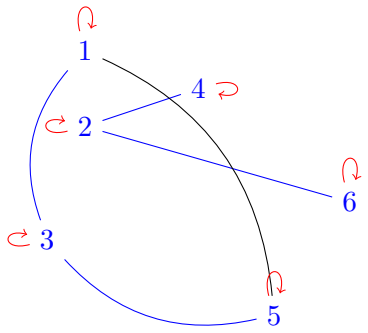
$$\forall x, y \in \mathcal{V}, \ \ ((x, y) \in \mathcal{C}) \wedge ((y, x) \in \mathcal{C}) \Rightarrow (x = y)\,.$$

La relación descrita en el Ejemplo 20 sí verifica la propiedad antisimétrica porque no se da ningún caso que verifique simultáneamente las dos hipótesis. Incluso si añadimos todas las refelxivas todo funciona bien. En el ejemplo de la circunferencia, sin embargo, no se da la antisimétrica: por ejemplo 1 y 0 verifican que $(1,0) \in \mathcal{C}$, $(0,1) \in R$ y, sin embargo, $1 \neq 0$.

- *Transitiva:* Que se expresa mediante:

$$\forall x, y, z \in \mathcal{V}, \quad ((x,y) \in \mathcal{C}) \wedge ((y,z) \in \mathcal{C}) \Rightarrow ((x,z) \in \mathcal{C}) .$$

La relación descrita en el Ejemplo 20 no verifica la transitiva. Tenemos que $(1,3) \in \mathcal{E}$ y $(3,5) \in \mathcal{E}$, pero $(1,5) \notin \mathcal{E}$. Tendríamos que añadirla para tener un grafo como:



Este último grafo ya nos dará una relación de orden. En el ejemplo de la circunferencia, sin embargo, se da la Transitiva, aunque no es relación de orden por no satisfacerse la anti–simétrica.

Relaciones de Equivalencia.

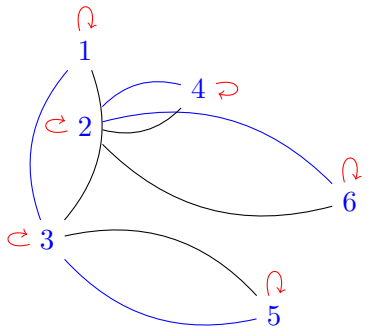
Son aquellas relaciones $\mathcal{C} \subseteq \mathcal{V} \times \mathcal{V}$, que verifican las propiedades siguientes:

- *Reflexiva:* (como en el caso anterior) $\forall x \in \mathcal{V}, \quad (x,x) \in \mathcal{C}$. En el Ejemplo 20 debemos completar nuestra lista de aristas, mientras que el ejemplo de la circunferencia ya la verifica.

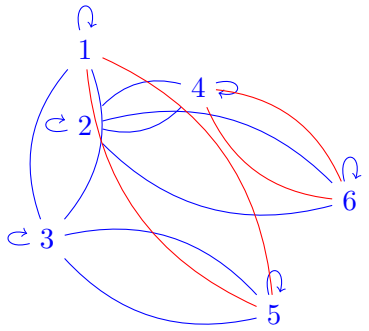
- *Simétrica:* $\forall x \in \mathcal{V}, \quad (x,y) \in \mathcal{C} \Leftrightarrow (y,x) \in \mathcal{C}$. En el caso de la circunferencia ya se satisface. Mientras que en el caso del Ejemplo 20 debems completar nuestra lista, añadiendo las aristas:

$$\{(3,1),(5,3),(4,2),(6,2)\},$$

para que se satisfaga. Esto nos da un grafo como:



- *Transitiva:* Que se expresa como ya se ha indicado. Es claro que el caso de la circunferencia tenemos una relación de equivalencia y en el caso del Ejemplo 20 habrá que completar con todos los casos. Esto nos dará una figura como la siguiente:



Este último grafo ya nos dará una relación de equivalencia.

Clasificando y Etiquetando elementos: Conjunto Cociente.

Dios crea, Carlos Linneo ordena

Sten Lindroth

Mientras las relaciones de orden pretenden establecer una preferencia de unos elementos sobre otros, dentro de un cierto conjunto, las relaciones de equivalencia pretenden clasificar los elementos del mismo conjunto para, posteriormente etiquetarlos. Se llamará conjunto cociente al conjunto de etiquetas finales.
El término etiqueta no es tan espúreo dado que las etiquetas son lo que definen, de manera bastante desafortunada en ocasiones, cosas tan dispares como la sociedad de consumo o la clasificación e Linneo de los seres vivos, por ejemplo.
Así, tomemos un conjunto $\mathcal{A}$ y una relación de equivalencia $\sim \subseteq \mathcal{A} \times \mathcal{A}$ definida sobre él. Para un elementos $a \in \mathcal{A}$, consideraremos su clase de equivalencia como el conjunto formado por todos los elementos de $\mathcal{A}$ equivalentes a a, es decir,

$$[a] := \{b \in \mathcal{A} \ : \ a \sim b\}.$$

Las clases son subconjuntos de $\mathcal{A}$ y se verifican las siguientes tres propiedades que indican que se trata de una partición de $\mathcal{A}$:

- $\mathcal{A} = \bigcup_{a \in \mathcal{A}} [a]$,
- $[[a] \cap [b] = \emptyset] \vee [[a] = [b]]$.
- $[a] \neq \emptyset$.

Así, retomando los ejemplos, podemos clasificar un colectivo $\mathcal{A}$ de personas (los habitantes de una ciudad, por ejemplo) mediante la relación de equivalencia *"$a \sim b$ si y solamente si [a tiene el mismo modelo de coche que b]"*. Se trata claramente de una relación de equivalencia sobre $\mathcal{A}$. La relación no es fina como clasificador puesto que hay individuos que poseen más de un coche y, por tanto, más de un modelo, con lo que podríamos tener que un "Dacia" y un BMW están relacionados. Admitamos que la relación se refina mediante *"$a \sim b$ si y solamente si [a e b poseen un mismo modelo de coche y ambos le prefieren entre los de su propiedad]"*.[2]
Ciertamente cada clase de euivalencia recorre todos los individuos de la una ciudad que poseen el mismo modelo de coche. Así, podríamos tener la clase [Luis], formada por todas las personas que no tienen coche o [Juan] formada por todas las personas que tienen un Dacia Logan del 96. De hecho, la etiqueta del coche define la clase. Podríamos usar el símbolo $\emptyset$ para describir la clase de quienes poseen ningún coche y el símbolo TT para quienes posean un Audi TT. Recíprocamente, en la sociedad de consumo, la publicidad no nos vende el coche que sale en un anuncio sino todos los coches equivalentes a él, es decir, todos los que tienen las mismas caracteísticas de los que fabrica esa empresa...Es lo que se llama "Marca" o "etiqueta" y es lo que los ciudadanos de las sociedades llamadas avanzadas compran.
En la clasificación de Linneo también tenemos una relación de equivalencia, esta vez entre todos los seres vivos. Dos seres vivos serían equivalentes si pertenecen al mismo Reino, Orden, Familia, Género, Especie....Luego se imponen las etiquetas. Así, la *rana muscosa* es la etiqueta que define la clase de equivalencia de todas las ranas de montaña de patas amarillas y no distingue entre ellas como individuos: una de tales ranas pertenece a la clase (etiqueta) pero ella no es toda la clase. En tiempos más recientes, el afán clasificatorio de Linneo se reconvierte en el afán clasificatorio de los genetistas: dos seres vivos son equivalentes si poseen el mismo sistema cromosósico (mapa genético), quedando el código genético como etiqueta individual.
Con ejemplos matemáticos, es obvio que en un grafo no orientado, las clases de equivalencia son las clausuras transitivas (o componentes conexas) de cada elemento. Otra forma de decirlo sería que dos vertices son equivalentes si desde uno se puede llegar a otro siguiendo aristas. En el caso de los número racionales, por ejemplo, la clase de equivalencia de 2/3 está formada por todos los pares de números enteros a/b, con $b \neq 0$, tales que $2b = 3a$.
Una vez queda claro que disponemos de clases de equivalencia, podemos considerarlas como elementos. Nace así el conjunto cociente que es el conjunto formado por las clases de equivalencia, es decir,

$$X/\sim := \{[x] \ : \ x \in X\}.$$

[2] Queremos hacer notar que los autores de estos apuntes no están en la misma clase de equivalencia.

En los ejemplos anteriores, el conjunto cociente es el conjunto de las etiquetas de coches, el conjunto de los nombres propuestos por Linneo para todas las especies de animales, etc. Nótese que el conjunto cociente es algo que, en muchas ocasiones, se puede escribir (por eso el término etiqueta) aunque hay casos en los que los conjuntos cocientes no son "etiquetables" en el sentido de formar un lenguaje. El ejemplo más inmediato es el caso de los números reales $\mathbb{R}$ que son las etiquetas de las clases de equivalencia de sucesiones de Cauchy, pero que no son expresables sobre un alfabeto finito.

6.5. Aplicaciones. Cardinales.

Las aplicaciones son un tipo particular de correspondencias.

Definición 118 (Aplicaciones) *Una aplicación entre dos conjuntos $\mathcal{A}$ y $\mathcal{B}$ es una correspondencia $\mathcal{C} \subseteq \mathcal{A} \times \mathcal{B}$ que verifica las propiedades siguientes:*

- *Todo elemento de $\mathcal{A}$ está realcionado con algún elemento de $\mathcal{B}$:*

$$\forall a \in \mathcal{A},\ \exists b \in \mathcal{B},\ (a,b) \in \mathcal{C}.$$

- *No hay más de un elemento de $\mathcal{A}$ que pueda estar relacionado con algún elemento de $\mathcal{B}$:*

$$\forall a \in \mathcal{A}, \forall b, c \in \mathcal{B},\ ((a,b) \in \mathcal{C}) \wedge ((a,c) \in \mathcal{C}) \Longleftarrow b = c.$$

En ocasiones se resume con la expresión:

$$\forall x \in A, \exists \mid y \in B,\ \ (x,y) \in R,$$

donde el cuantificador existencial modificado $\exists \mid$ significa "existe uno y sólo uno".

Esto es lo que aprendimos en el colegio, que una aplicación envia cada elemento a una única imagen. De esta forma se puede ver una aplicación como una transformación entre dos conjuntos.

Notación 119 *Notacionalmente se expresa $\mathcal{C} : \mathcal{A} \longrightarrow \mathcal{B}$, en lugar de $\mathcal{C} \subseteq \mathcal{A} \times \mathcal{B}$ y, de hecho, se suelen usar letras minúsculas del tipo $f : \mathcal{A} \longrightarrow \mathcal{B}$ para indicar la aplicación f de $\mathcal{A}$ en $\mathcal{B}$. Al único elemento de $\mathcal{B}$ relacionado con un $a \in \mathcal{A}$ se le representa $f(a)$ (es decir, escribimos $a \longmapsto f(a)$ en lugar de $(a, f(a)) \in f$).*
Por simplicidad, mantendremos la notación (inadecuada, pero unificadora) $f : \mathcal{A} \longrightarrow \mathcal{B}$ también para las correspondencias, indicando en cada caso si hacemos referencia a una aplicación o a una correspondencia, y reservaremos la notación $\mathcal{C} \subseteq \mathcal{A} \times \mathcal{A}$ para las relaciones.

Ejemplo 22 (Aplicación (o función) característica de un subsonjunto) *Sea $\mathcal{A}$ un conjunto $L \subseteq \mathcal{A}$ un subconjunto. De modo natural tenemos defínda una aplicación que toma como entradas los elementos de $\mathcal{A}$ y depende fuertemente de L: el la función característica*

$$\chi_L : \mathcal{A} \longrightarrow \{0,1\}$$

y que viene definida para cada $a \in \mathcal{A}$ mediante:

$$\chi_L(a) := \begin{cases} 1, & \text{si } a \in L, \\ 0, & \text{en otro caso-} \end{cases}$$

Se usa la expresión función cuando se trata de aplicaciones $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, expresión que viene de la tradición del Análisis Matemático.

Definición 120 (Composición) *Dados tres conjuntos $\mathcal{A}$, $\mathcal{B}$ y $\mathcal{C}$ y dos aplicaciones $f : \mathcal{A} \longrightarrow \mathcal{B}$, $h : \mathcal{B} \longrightarrow \mathcal{C}$, podemos definir una aplicación (llamada composición de f y g) que denotaremos $g \circ f : \mathcal{A} \longrightarrow \mathcal{C}$ y viene definida por la regla:*

$$a \longmapsto g(f(a)),\ \forall a \in \mathcal{A},$$

es decir, "primero aplicamos f sobre a y luego aplicamos g a $f(a)$".

Para una aplicación (o correspondencia) $f : \mathcal{A} \longrightarrow \mathcal{A}$ podemos definir la potencia mediante:

$$\begin{array}{ll} f^0 := & Id_A, \ \text{(la identidad)}, \\ f^1 := & f, \\ f^i := & f^{i-1} \circ f, \ \forall i \geq 2. \end{array}$$

Determinismo/Indeterminismo.

A partir de una aplicación (o correspondencia) $f : \mathcal{A} \longrightarrow \mathcal{A}$, podemos definir una estructura de grafo orientado "natural", definiendo los vértices como los elementos de $\mathcal{A}$ y las aristas mediante

$$\mathcal{V} := \{(a, f(a)) \ : \ a \in \mathcal{A}\}.$$

Dependiendo la formación del lector, puede ser que conoce a este conjunto de vértices por el nombre *"el grafo de la función f"*.
Dentro de ese grafo orientado, podemos considerar la parte de la "componente conexa" de a que son descendientes de a. Este conjunto vendrá dado por las iteraciones de f, es decir:

$$\{f^i(a) \mid i \in \mathbb{N}\}.$$

La diferencia entre el hecho de ser f aplicación o correspondencia se traduce en términos de "determinismo" o "indeterminismo":

- En el caso de ser aplicación, el conjunto de sucesores es un conjunto, posiblemente, infinito, en forma de camino (un árbol sin ramificaciones):

  $$a \longrightarrow f(a) \longrightarrow f^2(a) \longrightarrow f^3(a) \longrightarrow \cdots .$$

  Se dice que $(\mathcal{A}, f)$ tiene una dinámica *determinista*.

- En el caso de ser correspondencia, el conjunto de los sucesores de a toma la forma de árbol (con posibles ramificaciones): algunos valores no tendrán descendientes y otros tendrán más de un descendiente directo. Se dice que $(\mathcal{A}, f)$ tiene una dinámica *indeterminista*.

Ejemplo 23 *Tomemos* $\mathcal{A} := \mathbb{Z}/5\mathbb{Z} := \{0, 1, 2, 3, 4\}$*, las clases de restos módulo* 5 *y consideremos* $f := \mathcal{A} \longrightarrow \mathcal{A}$*, dada mediante:*

$$a \longmapsto f(a) = a^2, \forall a \in \mathcal{A}.$$

Es una aplicación, por lo que los descendientes de cada valor $a \in \mathcal{A}$ *forman un camino (un árbol sin ramificaciones). Por ejemplo,* $\{0\}$ *es el conjunto de todos los descendientes de* 0*, mientras que, si empezamos con* 3 *tendremos:*

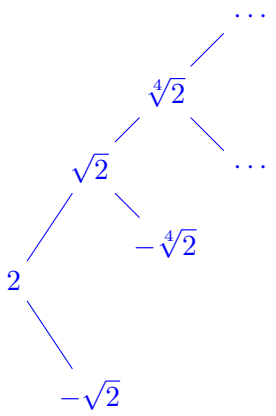
$$3 \longmapsto f(3) = 4 \longmapsto f^2(3) = 1 \longmapsto f^3(3) = 1 \longmapsto 1 \longmapsto \cdots,$$

Como f *es aplicación tendremos, para cada* $a \in \mathcal{A}$ *una dinámica determinista.*

Ejemplo 24 *Un ejemplo de indeterminismo sería* $\mathcal{A} = \mathbb{R}$ *y la correspondencia:*

$$\mathcal{C} := \{(x, y) \ : \ x = y^2\}.$$

En este caso, si $x < 0$ *no hay descendientes, si* $x = 0$ *hay solamente un deecendiente, y si* $x > 0$ *tenemos una infinidad de descendientes en forma de árbol no equilibrado. Por ejemplo,*



Los vértices $-\sqrt{2}, -\sqrt[4]{2}, \ldots$ *no tendrán descendientes, mientras los positivos tienen un par de descendientes directos.*
Debe señalarse que este ejemplo muestra un indeterminismo fuertemente regular (sabemos la regla) pero, en general, el indeterminismo podría presentar una dinámica muy impredecible.

Aplicaciones Biyectivas. Cardinales.

Sólo un pequeño resumen del proceso de contar el número de elementos de un conjunto, noción que preocupaba originalmente a G. Cantor. El entender el concepto de infinito también fue estudiado por el matemático David Hilbert en su paradoja del hotel con infinitas habitaciones. En esta paradoja, un hotel con infinitas habitaciones y sin habitaciones recibe varios huespedes que se quieren alojar en el hotel para la noche. El empleado del hotel, cambiando a los huespedes de habitación, consigue alojarlos a todos. Esta paradoja, que no es tal, se entiende utilizando aplicaciones.

Definición 121 (Aplicaciones inyectivas, suprayectivas, biyectivas) *Sea* $f : \mathcal{A} \longrightarrow \mathcal{B}$ *una aplicación.*

- *Decimos que* f *es inyectiva si verifica:*

$$\forall a, b \in \mathcal{A}, \quad (f(a) = f(b)) \Longrightarrow a = b.$$

- *Decimos que* f *es suprayectiva si verifica:*

$$\forall b \in \mathcal{B}, \exists a \in \mathcal{A}, \quad f(a) = b.$$

- *Decimos que* f *es biyectiva si es, a la vez, inyectiva y suprayectiva.*

Obsérvese que una aplicación $f : \mathcal{A} \longrightarrow \mathcal{B}$ es biyectiva si y solamente si disponemos de una aplicación (llamada inversa de f) que se suele denotar por $f^{-1} : \mathcal{B} \longrightarrow \mathcal{A}$ y que satisface:

$$f \circ f^{-1} = Id_B, \quad f^{-1} \circ f = Id_A,$$

donde $\circ$ es la composición y Id_A e Id_B son las respectivas aplicación identidad en $\mathcal{A}$ y en $\mathcal{B}$. En general las aplicaciones no tienen inversa, es decir, no podemos suponer siempre que sean biyectivas.
El proceso de contar no es sino la fundamentación del proceso "infantil" de contar mediante identificación de los dedos de las manos con los objetos a contar. Este proceso es una biyección.

Definición 122 (Cardinal) *Se dice que dos conjuntos* $\mathcal{A}$ *y* $\mathcal{B}$ *tienen el mismo cardinal (o número de elementos) si existe una biyección* $f : \mathcal{A} \longrightarrow \mathcal{B}$*. También se dice que son biyectables.*

- *Un conjunto* $\mathcal{A}$ *se dice finito si existe un número natural* $i \in \mathbb{N}$ *y una biyección*

$$f : \mathcal{A} \longrightarrow \{1, 2, 3, \ldots, i\}.$$

  Por abuso de lenguaje se identifican todos los conjuntos del mismo cardinal y escribiremos, en el caso finito, $\sharp(\mathcal{A}) = i$*, cuando* $\mathcal{A}$ *sea biyectable a* $\{1, 2, \ldots, i\}$*.*

- *Un conjunto* $\mathcal{A}$ *se dice (infinito) numerable si hay una biyección* $f : \mathcal{A} \longrightarrow \mathbb{N}$

- *Un conjunto se dice contable si es finito o numerable.*

Quizá, cuando pensemos en numerar, siempre pensemos en un sistema de numeración en base diez por el número de dedos que un humano medio posee. En esta definición, vemos que no hay ninguna precondición sobre el sistema de numeración, es más, podriamos definir el cardinal utilizando cualquier conjunto biyectable con $\{1, 2, \ldots, i\}$.
Esta es la primera propiedad, el cardinal es el mismo para cualquier conjunto biyectable,[3] ahora podemos demostrar otra propiedad referida al cardinal de dos conjuntos y las partes de esos conjuntos.

[3] notar que esto da una relación de equivalencia.

Proposición 123 *Si dos conjuntos $\mathcal{A}$ e $\mathcal{B}$ son biyectables, también son biyectables $\mathcal{P}(\mathcal{A})$ y $\mathcal{P}(\mathcal{B})$ (i.e. , las familias de sus subconjuntos).*

Demostración. Baste con disponer de una biyección $f : \mathcal{A} \longrightarrow \mathcal{B}$ para poder definir:

$$\widetilde{f} : \mathcal{P}(\mathcal{A}) \longrightarrow \mathcal{P}(\mathcal{B}),$$

dada mediante:

$$\mathcal{C} \longmapsto \widetilde{f}(\mathcal{C}) := \{\widetilde{f}(a) \in \mathcal{B} \ : \ a \in \mathcal{A}\} \subseteq \mathcal{B}.$$

La inversa de esta transformación será:

$$\widetilde{f}^{-1} : \{\mathcal{P}(\mathcal{B}) \longrightarrow \mathcal{P}(\mathcal{A})\},$$

dada mediante

$$\mathcal{C} \longmapsto \widetilde{f}^{-1}(\mathcal{C}) := \{a \in \mathcal{A} \ : \ f(a) \in \mathcal{B}\}.$$

Queda todavia por demostrar:

- $\widetilde{f}$ y $\widetilde{f}^{-1}$ son aplicaciones,
- una función es inversa de la otra.

La demostración se deja como ejercicio. ∎

Usualmente se utiliza la notación f y f^{-1} en lugar de $\widetilde{f}$ y $f^{-1}[w]$, usadas en la prueba anterior. Aunque son aplicaciones diferentes, ya que extienden de forma natural a la básica, se tiende a identificar una aplicación con otra.
Algunos cardinales y propiedades básicas:

a. Los conjuntos $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ son conjuntos numerables, mientras que $\mathbb{R}$ o $\mathbb{C}$ son conjuntos infinitos (no son finitos) y son no numerables (no son biyectables con $\mathbb{N}$).

b. Los subconjuntos de un conjunto finito son también finitos. Entre los subconjuntos $\mathcal{A}, \mathcal{B}$ de un conjunto finito se tiene la propiedad

$$\sharp(\mathcal{A} \cup \mathcal{B}) + \sharp(\mathcal{A} \cap \mathcal{B}) = \sharp(\mathcal{A}) + \sharp(\mathcal{B}).$$

Esta formula es un caso parcial de la formula de inclusión-exclusión.

c. Los subconjuntos de un conjunto contable son también contables. Para demostrar esto, es conveniente utilizar el principio de buena ordenación, que dice que todo subconjunto de $\mathbb{N}$ tiene un mínimo.

d. Si $\mathcal{A}$ y $\mathcal{B}$ son finitos tendremos:

$$\sharp\,(\mathcal{A} \times \mathcal{B}) = \sharp(\mathcal{A})\sharp(\mathcal{B}).$$

e. Si A es un conjunto finito, el cardinal de $\mathcal{P}(\mathcal{A})$ (el número de todos sus subconjuntos) es dado por

$$\sharp\,(\mathcal{P}(\mathcal{A})) = 2^{\sharp(\mathcal{A})}.$$

f. Si $\mathcal{A}$ es un conjunto finito, el número de aplicaciones $f : A \longrightarrow \{0, 1\}$ es $2^{\sharp(A)}$.

g. Si $\mathcal{A}$ es un conjunto finito,

$$\sharp(\mathcal{A}^n) = (\sharp(\mathcal{A}))^n .$$

Por ejemplo, si K es un cuerpo finito de la forma $K := \mathbb{Z}/p\mathbb{Z}$, donde $p \in \mathbb{N}$ es un número primo, el cardinal

$$\sharp(K^n) = \sharp(K)^n,$$

por lo que se tiene que para cada espacio vectorial V de dimensión finita sobre un cuerpo K finito se tiene:

$$\dim V = \log_{\sharp(K)} \sharp(V).$$

h. Si $\mathcal{A}$ es un conjunto finito $\sharp(\mathcal{A}) = n$, el número de permutaciones (es decir, biyecciones de $\mathcal{A}$ en sí mismo) es $n!$. Además, el número de subconjuntos de cardinal k de $\mathcal{A}$ es dado por el número combinatorio:

$$\binom{n}{k} := \frac{n!}{k!(n-k)!}.$$

De ahí que se tenga:

$$2^n := \sum_{k=0}^{n} \binom{n}{k}.$$

Algunas propiedades elementales de los cardinables contables se resumen en:

Proposición 124 *Productos finitos de conjuntos contables es un conjunto contable. Es decir, dados $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ una familia finita de conjuntos contables, entonces el producto cartesiano $\prod_{i=1}^{n} \mathcal{A}_i$ es también contable.*
La unión numerable de conjuntos contables es contable, es decir, dados $\{\mathcal{A}_i : i \in \mathbb{N}\}$ una familia numerable de conjuntos, de tal modo queo cada $\mathcal{A}_i$ es contable, entonces, tambiés es contable el conjunto:

$$\mathcal{A} := \bigcup_{i \in \mathbb{N}} \mathcal{A}_i.$$

Ejemplo 25 (Los subconjuntos de $\mathbb{N}$) *Por lo anterior, los subconjuntos de $\mathbb{N}$ son siempre conjuntos contables (finitos o numerables) pero la cantidad de subconjuntos de $\mathbb{N}$ es infinita no numerable (es decir, el cardinal de $\mathcal{P}(\mathbb{N})$ es infinito no numerable). Para comprobarlo, vamos a mostrar una biyección entre $\mathcal{P}(\mathbb{N})$ y el intervalo $[0, 1] \subseteq \mathbb{R}$ de números reales. Nótese que el cardinal del intervalo $[0, 1]$ es igual al cardinal de los números reales. Usaremos la función característica asociada a cada subconjunto $L \subseteq \mathbb{N}$. Así, dado $L \in \mathcal{P}(\mathbb{N})$, definiremos el número real:*

$$L \longmapsto x_L := \sum_{i=1}^{\infty} \frac{\chi_L(i)}{2^i} \in [0, 1].$$

Nótese que el número real asociado al conjunto vacío $\emptyset$ *es el número real* $x_\emptyset = 0$, *mientras que el número real* $x_{\mathbb{N}} \in [0,1]$ *es precisamente* $x_{\mathbb{N}} = 1 \in [0,1]$.
Recíprocamente, dado cualquier número real $x \in [0,1]$, *éste posee una única expansión "decimal" en base dos (para ser más correcto, digamos, una única expansión binaria):*

$$x := \sum_{i=1}^{\infty} \frac{x_i}{2^i}.$$

Definamos el sunconjunto $L_x \subseteq \mathbb{N}$ *mediante:*

$$L_x := \{i \in \mathbb{N} \ : \ x_i = 1\}.$$

Ambas aplicaciones ($x \longmapsto L_x$ *y* $L \longmapsto x_L$) *son una inversa de la otra y definen biyecciones entre* $[0,1]$ *y* $\mathcal{P}(\mathbb{N})$ *y recíprocamente).*
Dejamos al lector el esfuerzo de verificar que hay tantos número reales (en todo $\mathbb{R}$) *como número reales en el intervalo* $[0,1]$.

Bibliografía

[Aho–Hopcroft–Ullman, 75] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *The Design and Analysis of Computer Algotrithms.* Addison–Wesley (1975).

[Aho-Ullman, 72a] A. V. Aho, J. D. Ullman. *The Theory of Parsing, Translation and Compiling. Vol I: Parsing.* Prentice Hall, 1972.

[Aho-Ullman, 72b] A. V. Aho, J. D. Ullman. *The Theory of Parsing, Translation and Compiling. Vol II: Compilers.* Prentice Hall, 1972.

[Aho–Ullman, 95] A. V. Aho, J. D. Ullman. *Foundations of Computer Science.* W. H. Freeman (1995).

[Alfonseca, 07] M. Alfonseca. *Teoría De Autómatas y Lenguajes Formales.* McGraw–Hill, 2007.

[Balcázar–Díaz–Gabarró, 88] J. L. Balcazar, J. L. Díaz and J. Gabarró. *"Structural Complexity I"*, EATCS Mon. on Theor. Comp. Sci. **11**, Springer (1988).

[Balcázar–Díaz–Gabarró, 90] J. L. Balcázar, J. L. Díaz and J. Gabarró. *"Structural Complexity II"*, EATCS Mon. on Theor. Comp. Sci. Springer (1990).

[Chomsky, 57] N. Chomsky. *Syntactic Structures.* Mouton and Co. , The Hague, 1957.

[Chomsky–Miller, 57] N. Chomsky, G. A. Miller. "Finite state languages". *Information and Control* **1**:2 (1957) 91–112.

[Chomsky, 59a] N. Chomsky. "On certain formal properties of grammars". *Information and Control* **2**:2 (1959) 137–167.

[Chomsky, 59b] N. Chomsky. "A note on phrase structure grammars". *Information and Control* **2**: 4 (1959) 393–395.

[Chomsky, 62] N. Chomsky. "Context–free grammarsand pushdown storage". *Quarterly Progress Report* No. **65**. Research Laboratory of Electronics, M. I. T. , Cambridge, Mass. , 1962.

[Chomsky, 65] N. Chomsky. "Three models for the description of language". *IEEE Trans. on Information Theory* **2** (1965) 113-124.

[Cocke-Schwartz, 70] J. Cocke, J. T. Shwartz. *Programming Languages and their Compilers*. Courant Institute of Mathematical Sciences, NYU, 1970.

[RE Davis, 89] R. E. Davis. *" Truth, Deduction and Computation" (Logic and Semantics for Computer Science)"*. W. H. Freeman (1989).

[M Davis, 82] M. Davis. *"Computability and Unsolvability"* Dover (1982).

[M Davis, 97] M. Davis. "Unsolvable Problems". *Handbook of Mathematical Logic* North–Holland (1997) 567–594.

[Davis-Weyuker, 94] M. D. Davis, E. J. Weyuker. *"Computability, Complexity, and Languages (Fundamentals of Theoretical Computer Science), 2nd Ed. "*. Academic Press (1994).

[Eilenberg, 74] S. Eilenberg. *"Automata, Languages and Machines", vol. A*. Academica Press, Pure and App. Math. **59**–A (1974).

[Garey–Johnson, 79] M. R. Garey, D. S. Johnson. *"Computers and Intractability: A Guide to the Theory of NP–Completness"*. W. H. Freeman (1979).

[Hopcroft-Motwani–Ullman, 07] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3/Ed.* Addison-Wesley, 2007.

[Kleene, 52] S. S. Kleene. *Introduction to Metamathematics*. Van Nostrand Reinhold, New York, 1952.

[Kleene, 56] S. S. Kleene. "Representation of events in nerve nets". In *Automata Studies*. Shannon and McCarthy eds. Princeton University Press, Princeton, N. J. (1956) 3–40.

[Knuth, 65] D. E. Knuth. "On the Translation of Languages from Left to Right". *Information Control* **8** (1965) 707–639.

[Knuth, 97–98] D. E. Knuth. *"The art of computer programming (2nd Ed.), vol. 2 Seminumerical Algorithms"*. Addison–Wesley (1997–98).

[Kozen, 92] D. C. Kozen. *"The Design and Analysis of Algorithms"*. Texts and Monographs in Computer Science, Springer Verlag (1992).

[HandBook, 92] J. van Leeuwen (ed.). *Handbook of Theoretical Computer Science* Elsevier, 1992.

[Lewis-Papa, 81] H. L. Lewis, C. H. Papadimitriou. *"Elements of the Theory of Computation"*. Prentice–Hall (1981).

[Martin, 03] J. Martin. *"Introduction to Languages and the Theory of Computation, 3rd Edition"*. McGraw Hill, 2003.

[Marcus, 67] S. Marcus. *"Algebraic Linguistics; Analytic Models"*. Mathematics in Science and Engineering, vol. **29**, Academic Press (1967).

[Martin-Mitrana-Paun, 04] C. Martin-Vide, V. Mitrana, G. Paun. *"Formal Languages and Applications".* Springer, 2004.

[Papadimitriou, 94] C. H. Papadimitrou. *Computational Complexity.* Addison–Wesley (1994)

[Savitch, 70] W. J. Savitch. "Relationships between nondeterministic and deterministic tape complexities". *J. Comput. System. Sci.* **4** (1970) 177–192.

[Schönhage–Vetter, 94] A. Schönhage, E. Vetter. *"Fast Algorithms. A Multitape Turing machine Implementation".* Wissenschaftverlag (1994).

[Sipser, 97] M. Sipser (1997). *"Introduction to the Theory of Computation".* PWS Publishing (1997).

[Wagner–Wechsung, 86] K. Wagner, G. Wechsung. " *Computational complexity*" . D. Reidel (1986).

[Weihrauch, 97] K. Weihrauch. *"Computability".* EATCS monographs on Theor. Comp. Sci. **9**, Springer Verlag (1987).

[Wirth, 96] N. Wirth, *Compiler Construction.* Addison–Wesley International Computer SCience Service, 1996.