Teoría de Autómatas, Lenguajes Formales y Gramáticas

David Castro Esteban

Universidad de Alcalá

This is as near, I take it,
as a finite mind can ever come
to "perceiving everything that
is happening everywhere in the
universe".

The Doors of Perception
Aldous Huxley

Índice general

Introducción

Estas notas pretenden servir de guía en una asignatura sobre Teoría de Autómatas. En el caso particular de la Universidad de Alcalá, asignaturas: "Autómatas, lenguajes formales y gramáticas I y II" de la Ingeniería en Informática o "Teoría de Autómatas y Lenguajes Formales" de la Ingeniería Técnica en Informática de Sistemas pueden articularse entorno a los mismos. No pretenden, en medida alguna, suplantar los excelentes textos existentes acerca de los temas aquí tratados.

Lo que sigue es un breve resumen de los contenidos de los mismos.

Capítulo 1.- Autómatas finitos deterministas, autómatas finitos no deterministas y con transiciones vacías, equivalencias entre los distintos tipos de autómatas finitos.

Capítulo 2.- Gramáticas regulares, expresiones regulares, equivalencia entre autómatas finitos y gramáticas o expresiones regulares, lenguajes regulares.

Capítulo 3.- Lema de Bombeo (lenguajes regulares), propiedades de clausura, propiedades de decisión, equivalencia y minimización de autómatas finitos.

Capítulo 4.- Autómatas a pila, autómatas a pila deterministas.

Capítulo 5.- Gramáticas independientes del contexto, árboles de derivación, ambiguedad en gramáticas y lenguajes, equivalencia entre autómatas a pila y gramáticas independientes del contexto.

Capítulo 6.- Formas normales para gramáticas independientes del contexto, Lema de Bombeo (lenguajes independientes del contexto), existencia de lenguajes no independientes del contexto, propiedades de clausura y decisión de los lenguajes independientes del contexto.

Capítulo 7.- Máquinas de Turing, introducción a la Teoría de la Complejidad (Conjetura de Cook), Tesis de Church-Turing.

Capítulo 9.- Lenguaje recursivo y recursivamente enumerables, máquina de Turing universal y ejemplos de lenguajes no recursivos y no recursivamente enumerables, problemas indecidibles, reducciones, propiedades de los lenguajes recursivamente enumerables (Teorema de Rice).

Capítulo 8.- Gramáticas generales y sensibles al contexto, autómatas linealmente acotados, equivalencia con máquinas de Turing de las gramáticas generales, equivalencia con lenguajes sensibles al contexto de los autómatas linealmente acotados, Teorema de la Jerarquía de Chomsky.

Capítulo 10.- Funciones recursivas primitivas, recursivas y recursivas parciales, equivalencia con máquinas de Turing.

A parte de los capítulos anteriores, se ha incluido un apéndice destinado a las máquinas RAM. En él, se introduce este modelo alternativo de computación y se establece su equivalencia con el modelo de máquinas de Turing.

Alcalá de Henares, 17 de marzo de 2005.
David Castro Esteban

Capítulo 1

Autómatas finitos

En este Capítulo se presenta el modelo de autómata finito, discutiéndose algunos detalles del mismo. En la Sección 1.1 se define el concepto de autómata finito determinista y en la Sección 1.2 se introducen los autómatas finitos no deterministas y los autómatas finitos con transiciones vacías, mostrándose su equivalencia con los deterministas.

La referencia básica a lo largo del presente Capítulo es [7]. En ella se aborda el tema de manera intuitiva y clara. El lector interesado en una referencia más técnica puede acudir a la edición anterior, i.e. [6], o a la monografía [9].

1.1. Autómatas finitos deterministas

Intuitivamente, un autómata es un dispositivo teórico capaz de procesar una secuencia finita de símbolos de un alfabeto, cambiando su estado, si procede. De entre los posibles estados que puede alcanzar el autómata, se distinguen los *aceptadores* que indican el reconocimiento, por parte del autómata, de la secuencia tratada.

La definición formal es la siguiente:

Definición 1 (Autómata finito determinista, AFD) *Un autómata finito determinista es una tupla* $D = (K, \Sigma, \delta, q_0, F)$ *donde:*

- K *es un conjunto (finito) de estados,*
- Σ *es una alfabeto (finito),*
- $\delta : K \times \Sigma \longrightarrow K$ *es la función de transición,*
- $q_0 \in K$ *es el estado inicial, y*

- *$F \subseteq K$ es el conjunto de los estados aceptadores.*

Obviamente, la definición anterior es *estática*, no determina el funcionamiento del autómata sino que indica los elementos que lo conforman. Para dotar de dinamismo a la noción, extendemos el dominio de definición de la función δ a $K \times \Sigma^*$.

Definición 2 (Función de transición extendida, AFD) *Dado un autómata finito determinista $D = (K, \Sigma, \delta, q_0, F)$, se define la extensión de δ al dominio $K \times \Sigma^*$ (denotada por $\hat{\delta}$) de manera inductiva en la longitud de las cadenas:*

- *dado $q \in K$, definimos $\hat{\delta}(q, \varepsilon) := q$,*
- *dados $q \in K$, $\omega \in \Sigma^*$ y $\sigma \in \Sigma$, definimos $\hat{\delta}(q, \omega\sigma) := \delta(\hat{\delta}(q, \omega), \sigma)$.*

La definición anterior no es más que la formalización del funcionamiento de un autómata: dada una cadena $\sigma_1 \cdots \sigma_n$, el autómata lee cada uno de los símbolos de entrada, de izquierda a derecha, y cambia su estado, si procede, en función del estado y el símbolo leído. Las cadenas que llevan al autómata a un estado aceptador conforman el lenguaje del autómata. Formalmente:

Definición 3 (Lenguaje de un AFD/Lenguaje regular) *Dado un autómata finito determinista $D = (K, \Sigma, \delta, q_0, F)$, se define el lenguaje aceptado por el autómata, denotado $L(D)$, como:*

$$L(D) := \{\omega \in \Sigma^* : \hat{\delta}(q_0, \omega) \in F\}.$$

Los lenguajes que son aceptado por un autómata finito determinista se llaman regulares.

Típicamente, para describir un autómata basta detallar cada uno de sus elementos. En concreto, la función de transición δ suele representarse mediante una tabla, llamada de transiciones, que indica para cada par estado–símbolo el estado indicado por δ.

Ejemplo 4 *Considerar, por ejemplo, el autómata $A = (K, \Sigma, \delta, q_0, F)$, donde $K := \{q_0, q_1, q_2\}$, $\Sigma := \{0, 1\}$, $F := \{q_1\}$ y la función de transición δ se describe mediante la tabla de transiciones:*

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

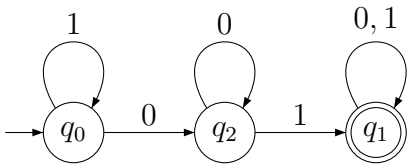
La flecha indica el estado inicial mientras que el asterisco, indica el estado final.

Es fácil ver que el autómata anterior acepta las cadenas que contienen la secuencia 01.

Una representación alternativa de los autómatas es la dada por los diagramas de transición. En ella se representa el autómata mediante un grafo orientado sujeto a las siguientes consideraciones:

- cada nodo del grafo tiene asociado de manera unívoca un estado del autómata,
- dos nodos, digamos p y q, están unidos mediante un arco dirigido de p a q si y sólo si existe un símbolo del alfabeto, digamos $\sigma \in \Sigma$, tal que $\delta(p, \sigma) = q$; dicho arco se etiqueta con el símbolo σ; si existen diversos de tales símbolos, digamos $\sigma_1, \ldots, \sigma_n \in \Sigma$, el arco se etiqueta con $\sigma_1, \ldots, \sigma_n$,
- los nodos correspondientes a estados aceptadores (finalizadores) se representan mediante una doble circunferencia con el fin de distinguirlos de los restantes, y
- al nodo correspondiente al estado inicial llega una flecha sin origen concreto.

Ejemplo 5 *La representación, vía diagrama de transición, del autómata finito determinista del Ejemplo 4 es como sigue:*



1.2. Autómatas finitos no deterministas

En la Sección anterior se han tratado los autómatas finitos deterministas. En dichos autómatas el cambio de estado está fijado de manera unívoca por el estado en el que se encuentran y el carácter que estan procesando. Si eliminamos dicha restricción obtenemos los conocidos como autómatas no deterministas.

Definición 6 (Autómata finito no determinista, AFN) *Un autómata finito no determinista es una tupla $N = (Q, \Sigma, \delta, q_0, F)$ donde:*

- *K es un conjunto (finito) de estados,*
- *Σ es una alfabeto (finito),*
- *$\delta : K \times \Sigma \longrightarrow \mathcal{P}(K)$ es la función de transición,*[1]
- *$q_0 \in K$ es el estado inicial, y*
- *$F \subseteq K$ es el conjunto de los estados aceptadores.*

Al igual que en el caso de los autómatas finitos deterministas, con el fin de detallar su funcionamiento, debemos extender el dominio de la función de transición a $K \times \Sigma^*$.

Definición 7 (Función de transición extendida, AFN) *Dado un autómata finito no determinista $N = (K, \Sigma, \delta, q_0, F)$, se define la extensión de δ al dominio $K \times \Sigma^*$ (denotada por $\hat{\delta}$) de manera inductiva en la longitud de las cadenas:*

- *dado $q \in K$, definimos $\hat{\delta}(q, \varepsilon) := \{q\}$,*
- *dados $q \in K$, $\omega \in \Sigma^*$ y $\sigma \in \Sigma$, definimos*

$$\hat{\delta}(q, \omega\sigma) := \bigcup_{p \in \hat{\delta}(q,\omega)} \delta(p, \sigma).$$

En cuanto al lenguaje aceptado por un autómata finito no determinista, se define como sigue.

Definición 8 (Lenguaje de un AFN) *Dado un autómata finito no determinista $N = (K, \Sigma, \delta, q_0, F)$, definimos el lenguaje aceptado por el autómata, denotado $L(N)$, como:*

$$L(N) := \{\omega \in \Sigma^* : \hat{\delta}(q_0, \omega) \cap F \neq \emptyset\}.$$

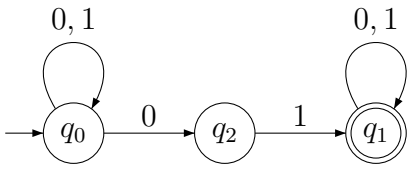
Retomando el Ejemplo 4, veamos un autómata finito no determinista que reconoce el mismo lenguaje:

[1] $\mathcal{P}(K)$ denota el conjunto de todos los subconjuntos de K, esto es, las partes de K.

Ejemplo 9 *Considerar el autómata $N = (K, \Sigma, \delta, q_0, F)$, donde el conjunto de estados es $K := \{q_0, q_1, q_2\}$, el alfabeto $\Sigma := \{0, 1\}$, el conjunto de estados aceptadores es $F := \{q_1\}$ y la función de transición δ se describe mediante la tabla de transiciones:*

	0	1
$\rightarrow q_0$	$\{q_0, q_2\}$	$\{q_0\}$
$*q_1$	$\{q_1\}$	$\{q_1\}$
q_2	$\emptyset$	$\{q_1\}$

La representación mediante diagrama de transición es como sigue:



Obviamente, el lenguaje aceptado es el mismo que en el Ejemplo 4.

El ejemplo anterior ilustra el hecho de que todo lenguaje aceptado por una autómata finito determinista, es aceptado por una autómata finito no determinista. De hecho, todo autómata finito determinista $D = (K, \Sigma, \delta, q_0, F)$ puede interpretarse como un autómata finito no determinista, digamos $N = (K', \Sigma, \delta', q_0', F')$, sin más que definir $K' := K$, $q_0' := \{q_0\}$, $F' := F$ y

$$\begin{aligned} \delta' : \quad K' \times \Sigma \quad &\rightarrow \mathcal{P}(K) \\ (q, \omega) \quad &\mapsto \{\delta(q, \omega)\}. \end{aligned}$$

El recíproco también se verifica, si bien es cierto que su demostración resulta más complicada.

Teorema 10 (Equivalencia entre AFD y AFN) *Todo lenguaje $L \subseteq \Sigma^*$ aceptado por un autómata finito no determinista es aceptado por un autómata finito determinista.*

Demostración .– Sea $L \subseteq \Sigma^*$ un lenguaje aceptado por un autómata finito no determinista $N = (K, \Sigma, \delta, q_0, F)$. En lo que sigue, detallaremos la construcción de un autómata finito determinista $D = (K', \Sigma, \delta', q_0', F')$ que acepte L (simule a N).
Definimos:

- el conjunto de estados de D mediante $K' := \mathcal{P}(K)$,
- el conjunto de estados aceptadores por

$$F' := \{q \in K' : q \cap K \neq \emptyset\},$$

- la función de transición δ' como

$$\delta'(q, \sigma) := \delta(q \times \{a\}) = \cup_{p \in q} \delta(p, \sigma), \text{ y}$$

- el estado inicial mediante $q'_0 := \{q_0\}$.

Queda demostrar que el autómata $D = (K', \Sigma, \delta', q'_0, F')$ acepta el mismo lenguaje que N. Para ello, demostraremos que para toda cadena $w \in \Sigma^*$ se satisface $\hat{\delta}'(q'_0, w) = \hat{\delta}(q_0, w)$.
El caso base resulta trivial puesto que $w = \varepsilon$ y $q'_0 = \{q_0\}$. Supongamos probado el resultado para todas las cadenas de longitud n y veamos que se verifica para las de longitud $n + 1$.
Dado $w \in \Sigma^*$ un palabra de longitud n y $\sigma \in \Sigma$ un elemento del alfabeto, se verifica

$$\hat{\delta}'(q'_0, w\sigma) := \delta'(\hat{\delta}'(q'_0, w), \sigma) = \delta'(\hat{\delta}(q_0, w), \sigma).$$

Por lo tanto, de acuerdo a la definición de δ', se sigue

$$\delta'(\hat{\delta}(q_0, w), \sigma) = \bigcup_{q \in \hat{\delta}(q_0, w)} \delta'(q, \sigma)$$

■

Queremos insistir en el hecho de que la demostración anterior es constructiva, i.e. describe un algoritmo que construye, a partir de un autómata finito no determinista, un autómata finito determinista cuyo lenguaje es el mismo.

Hasta la fecha, los autómatas cambian de estado al procesar símbolos de la entrada. Permitir que los autómatas cambien de estado sin procesar símbolos de la entrada da lugar a la noción de autómata no determinista con transiciones vacías.

Definición 11 (Autómata finito con transiciones ε, ε–AFN) *Un autómata finito con transiciones ε es una tupla $E = (K, \Sigma, \delta, q_0, F)$ donde:*

- *K es un conjunto (finito) de estados,*
- *Σ es una alfabeto (finito),*

- $\delta : K \times (\Sigma \cup \{\varepsilon\}) \longrightarrow \mathcal{P}(K)$ *es la función de transición,*
- $q_0 \in K$ *es el estado inicial, y*
- $F \subseteq K$ *es el conjunto de los estados aceptadores.*

Al admitir transiciones vacías, el autómata puede cambiar de estado sin procesar carácter alguno. Por lo tanto, para precisar como evoluciona el autómata, hay que introducir el concepto de clausura con respecto a ε.

Definición 12 (Clausura respecto de ε) *Dado un autómata finito no determinista con transiciones vacías $E = (K, \Sigma, \delta, q_0, F)$, se define la clausura de un estado $q \in K$, con respecto de ε, de manera recursiva*

- $q \in \mathrm{Cl}_\varepsilon(q)$,
- *para todo $p \in \mathrm{Cl}_\varepsilon(q)$ y todo $r \in Q$ tales que $r \in \delta(p, \varepsilon)$, se verifica $r \in \mathrm{Cl}_\varepsilon(q)$.*

En general, dado un conjunto $Q \subseteq K$, se define la clausura de Q con respecto a transiciones vacías de manera análoga.

Intuitivamente, la clausura con respecto a ε de un estado es el conjunto de estados a los que puede llegar el autómata sin procesar símbolo alguno. Teniendo en cuenta esta noción, se define la función de transición extendida de la siguiente manera:

Definición 13 (Función de transición extendida, AFN–ε) *Dado un autómata finito no determinista con transiciones vacías $E = (K, \Sigma, \delta, q_0, F)$, se define la extensión de δ al dominio $K \times \Sigma^*$ (denotada por $\hat{\delta}$) de manera inductiva en la longitud de las cadenas:*

- *dado $q \in K$, definimos $\hat{\delta}(q, \varepsilon) := \mathrm{Cl}_\varepsilon(q)$,*
- *dados $q \in K$, $\omega \in \Sigma^*$ y $\sigma \in \Sigma$, sea*

$$P := \bigcup_{p \in \hat{\delta}(q,\omega)} \delta(p, \sigma),$$

definimos

$$\hat{\delta}(q, \omega\sigma) := \bigcup_{p \in P} \mathrm{Cl}_\varepsilon(p).$$

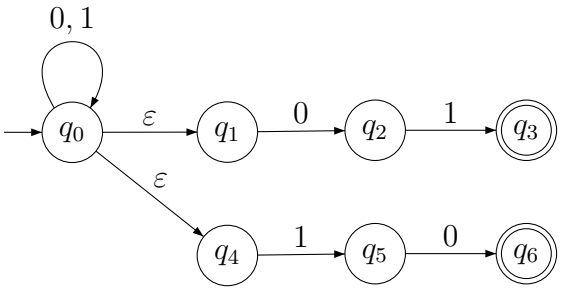
En cuanto al lenguaje aceptado por un autómata finito no determinista con transiciones vacías, se define como sigue.

Definición 14 (Lenguaje aceptado por un AFN–ε) *Dado un autómata finito no determinista con transiciones vacías $E = (K, \Sigma, \delta, q_0, F)$, se define el lenguaje aceptado por el autómata, denotado $L(E)$, como:*

$$L(E) := \{\omega \in \Sigma^* : \hat{\delta}(q_0, \omega) \cap F \neq \emptyset\}.$$

Veamos un ejemplo sencillo de autómata no determinista con transiciones vacías.

Ejemplo 15 *Sea $E = (K, \Sigma, \delta, F, q_0)$ es autómata finito no determinista con transiciones vacías descrito por el siguiente diagrama de transición:*



El lenguaje que acepta este autómata es el de las cadenas que contienen la secuencia 01*. En este caso, por ejemplo, $Cl_\varepsilon(q_0) = \{q_0, q_1\}$ mientras que $Cl_\varepsilon(q_1) = \{q_1\}$.*

Obviamente, un autómata determinista puede interpretarse como un autómata no determinista con transiciones vacías sin más que definir $\delta(q, \varepsilon) = \emptyset$. El siguiente resultado demuestra el recíproco.

Teorema 16 (Equivalencia entre AFD y AFN–ε) *Todo lenguaje $L \subseteq \Sigma^*$ aceptado por un autómata finito no determinista con transiciones vacías es aceptado por un autómata finito determinista.*

Demostración .– La demostración de este resultado es similar al de la equivalencia entre autómatas deterministas y no deterministas.
Supongamos dado un autómata con transiciones vacías $E = (K, \Sigma, \delta, q_0, F)$. Construyamos un autómata finito determinista $A = (K', \Sigma, \delta', q_0, F)$ que acepte el mismo lenguaje. Para ello definimos

- el conjunto de estados mediante $K' := \{P \subseteq K : \mathrm{Cl}_\varepsilon(P) = P\}$, esto es, Q' es el conjunto de subconjunto de K cerrados bajo clausura con respecto a ε.
- el estado inicial por $q'_0 := \mathrm{Cl}_\varepsilon(q_0)$,
- el conjunto de estados aceptadores mediante

$$F' := \{P \in K' : P \cap F \neq \emptyset\}, \text{ y}$$

- dado $P \in K'$ y $\sigma \in \Sigma$ y siendo

$$S := \bigcup_{p \in P} \delta(p, \sigma),$$

definimos

$$\delta'(P, \sigma) := \bigcup_{s \in S} \mathrm{Cl}_\varepsilon(s).$$

Veamos que el autómata finito determinista así construido acepta exactamente las mismas cadenas que E. Para ello, demostremos por inducción (en la longitud de las cadenas) que $\hat{\delta}(q_0, \omega) = \hat{\delta}'(q'_0, \omega)$.
En primer lugar veamos el caso base, i.e. $\omega = \varepsilon$. Por definición de $\hat{\delta}'$ tenemos

$$\hat{\delta}'(q'_0, \varepsilon) = q'_0 = \mathrm{Cl}_\varepsilon(q_0) = \hat{\delta}(q_0, \varepsilon).$$

Suponiendo cierta la afirmación para cadenas de longitud $n \in \mathbb{N}$, sea $\omega \in \Sigma^*$ una cadena de longitud n y sea $\sigma \in \Sigma$ un símbolo del alfabeto. Por hipótesis tenemos que

$$S = \hat{\delta}(q_0, \omega) = \hat{\delta}'(q'_0, \omega).$$

Ahora bien, $\hat{\delta}(q_0, \omega\sigma)$ se define como

$$\hat{\delta}(q_0, \omega\sigma) := \bigcup_{s \in S} \mathrm{Cl}_\varepsilon(s),$$

por lo que coincide con $\hat{\delta}'$ por la definición de ésta. ∎

Al igual que en el caso del Teorema de equivalencia entre AFD y AFN, la demostración del resultado anterior es constructiva al dar un algoritmo de construcción de un autómata finito determinista a partir de un autómata finito no determinista con transiciones vacías.

Capítulo 2

Gramáticas regulares

En este Capítulo se presentan las nociones de gramáticas y expresiones regulares. Hecho esto, se muetran sus relaciones con los autómatas finitos: se demuestra que esencialmente son nociones equivalentes tanto en cuanto permiten describir los mismos objetos, a saber: los lenguajes regulares.

2.1. Expresiones y gramáticas regulares

A lo largo de está Sección damos las definiciones de expresión y gramática regular, así como diversos ejemplos. Coloquialmente hablando, tanto las expresiones regulares como las gramáticas son distintas formas de especificar los lenguajes regulares (como se verá en Secciones subsiguientes).

La definición formal de expresión regular es como sigue:

Definición 17 (Expresión regular) *Dado un alfabeto finito Σ, se define un expresión regular, vía inducción estructural, de la siguiente manera:*

- *(Base) $\emptyset$,ε y σ (con $\sigma \in \Sigma$) son expresiones regulares cuyos lenguajes asociados son $\emptyset$, $\{\varepsilon\}$ y $\{\sigma\}$ respectivamente.*
- *(Inducción) Distinguimos los siguientes casos:*
  - *Dada una expresión regular E, (E) es un expresión regular cuyo lenguaje es precisamente $L(E)$ (el lenguaje asociado a E).*
  - *Dada un expresión regular E, E^* es un expresión regular cuyo lenguaje es $L(E)^*$ (la cláusura de $L(E)$ con respecto a la concatenación).*
  - *Dadas dos expresiones regulares E_1 y E_2, $E_1 + E_2$ es un expresión regular cuyo lenguaje es $L(E_1) \cup L(E_2)$ (la unión de los lenguajes asociados a E_1 y E_2).*

- *Dadas dos expresiones regulares E_1 y E_2, E_1E_2 es una expresión regular cuyo lenguaje asociado es $L(E_1)L(E_2)$ (la concatenación de los lenguajes de E_1 y E_2).*

Ejemplo 18 *Un ejemplo sencillo de expresión regular es, por ejemplo, el siguiente:*

$$(0+1)^*01(0+1)^*.$$

Dicha expresión describe el conjunto de las cadenas de ceros y unos que continen la subcadena 01*. Para verlo, basta observar que $(0+1)^*$ describe cualquier cadena de ceros y unos.*

En cuanto a la definición de gramática regular, es como sigue.

Definición 19 (Gramática regular (lineal a derecha)) *Una gramática regular (lineal a derecha) es una tupla $G = (V, T, P, S)$ donde*

- *V es un conjunto finito de símbolos llamados variables o símbolos no terminales,*
- *T es un conjunto finito de símbolos, llamados terminales, satisfaciendo $T \cap V = \emptyset$,*
- *$P \subseteq V \times (T^*V \cup T)$ es un conjunto finito de elementos, llamados producciones, de manera tal que si $(p, q) \in P$ se denotará mediante la expresión $p \to q$; p recibe el nombre de cabeza y q el de cuerpo de la producción, y*
- *$S \in V$ es el símbolo inicial.*

La definición anterior, tal y como se verá en los ejercicios, puede darse en términos de gramáticas lineales a izquirda. Afortunadamente, dicha elección no tiene relevancia alguna puesto que son equivalentes.

Un ejemplo sencillo de gramática es el siguiente:

Ejemplo 20 *Continuando con el Ejemplo 18, la gramática que describe el lenguaje asociado a $(0+1)^*01(0+1)^*$ viene dada por las producciones siguientes:*

$$S \to 0S$$

$$S \to 1S$$

$$S \to 01S'$$

$$S' \to 0S'$$

$$S' \to 1S', \ y$$

$$S' \to \varepsilon.$$

Más compactamente,

$$S \to 0S|1S|01S'$$

$$S' \to 0S|1S|\varepsilon.$$

Cabe destacar que en la definición de expresión regular hemos definido, de paso, el lenguaje asociado a la misma. Sin embargo, en la definición de gramática no hemos dado un tal lenguaje. Para asociar un lenguaje a una gramática debemos, en primera instancia, definir el concepto de derivación.

Definición 21 (Derivación) *Dada una gramática regular $G = (V, T, P, S)$ y una cadena αv, con $\alpha \in T^*$ y $v \in V$, diremos que $\alpha\gamma \in (T^*V \cup T^*)$ deriva de αv, si existe una producción $v \to \gamma \in P$. Si es el caso, lo indicaremos mediante la notación $\alpha v \Rightarrow_G \alpha\gamma$; en general, si una cierta cadena $\beta \in (T^*V \cup T^*)$ deriva de $\delta \in (T^*V \cup T^*)$ tras un número finito de derivaciones, lo notaremos mediante $\delta \Rightarrow_G^* \gamma$.*

Subyacente a la noción de derivación se encuentra el concepto de lenguaje asociado a una gramática regular en los siguientes términos:

Definición 22 (Lenguajes de una gramática regular) *Dada una gramática regular $G = (V, T, P, S)$, se defina el lenguaje de G como el conjunto de las cadenas de T^* derivables en un número finito de pasos a partir de S, i.e.*

$$L(G) = \{w \in T^* : S \Rightarrow_G^* w\}.$$

Continuando con el Ejemplo 20.

Ejemplo 23 *La gramática definida en el Ejemplo 20 da el lenguaje de las cadenas de ceros y unos que contienen la subcadena* 01.

Ejemplo 24 $(0 + 1)^*01(0 + 1)^*$ *es una expresión regular cuyo lenguaje es, precisamente, el lenguaje de las cadenas de ceros y unos que contienen la subcadena* 01.

2.2. Autómatas finitos y gramáticas regulares

A lo largo de esta Sección demostraremos que los lenguajes descritos vía gramáticas regulares coinciden con los lenguajes aceptados por autómatas finitos. Para ello, veamos de entrada que, dado un autómata, existe una gramática regular que describe el lenguaje aceptado por éste.

Teorema 25 *Sea A un autómata finito. Entonces, existe una gramática regular G tal que $L(G) = L(A)$.*

Demostración .– Supongamos de entrada que el autómata $A = (K, \Sigma, \delta, q_0, F)$ no tiene a q_0 por estado aceptador. Construimos la gramática G tal que $L(G) = L(A)$ como sigue: el conjunto de variables de G coincide con Q (el conjunto de estados de A), el conjunto de carácteres terminales de G es Σ y su símbolo inicial es q_0. Las producciones de G se definen como sigue: dada una variable $q \in Q$, $q \to \sigma p$ (con $p \in Q \setminus F$ y $\sigma \in \Sigma$) es una producción si $\delta(q, \sigma) = p$; dada una variable $q' \in Q$, $q \to \sigma$ (con $\sigma \in \Sigma$) es una producción si $\delta(q, \sigma) \in F$.
Queda claro que una cierta cadena $w \in \Sigma^*$ se deriva del símbolo inicial de G si y sólo si es aceptado por el autómata. Por lo tanto, $L(G) = L(A)$.
En el caso de que el estado inicial de A sea también un estado aceptador, basta introducir un nuevo símbolo inicial en la gramática, digamos S, y las producciones $S \to q_0$ y $S \to \varepsilon$. ∎

TODO 1 ***Ejemplo 26 (Conversión de AFD a G)***

Recíprocamente, veamos que dada una gramática regular, existe un autómata que acepta el lenguaje dado por ésta.

Teorema 27 *Sea G una gramática regular. Entonces, existe un autómata finito (no determinista) A tal que $L(A) = L(G)$.*

Demostración .– Supongamos dada una gramática regular $G = (V, T, P, S)$ (lineal a derecha). Construimos el autómata $A = (K, T, \delta, q_0, F)$ como sigue:

- el conjunto de estado viene dado por las expresiones de la forma $[v]$, donde v es una subcadena del cuerpo de alguna producción,
- el alfabeto del autómata coincide con el conjunto de caracteres terminales de G,

- la función de transición se define mediante las reglas siguientes: dado un estado $[p]$, con p una variable de G, y un símbolo $t \in T$, se define

$$\delta([p], t) = \{[q] : p \to q \in P\}.$$

Dados $t \in T$ y $p \in T^* \cup T^*V$ definiendo un estado $[p]$, se define

$$\delta([tp], t) = \{[p]\}$$

Es fácil ver, por inducción en la longitud de las cadenas, que $L(A) = L(G)$. ■

TODO 2 ***Ejemplo 28 (Conversión de G a AFD)***

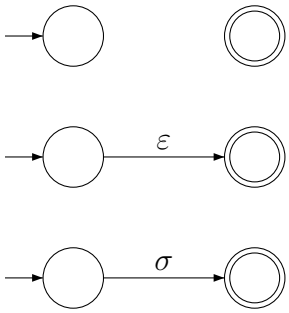
En resumen, hemos visto que tanto los autómatas como las gramáticas regulares definen los mismos lenguajes: los lenguajes regulares. Ambos constituyen una representación finita de objetos potencialmente infinitos (tanto las gramaticas como los autómatas son objetos finitos, descritos de manera finita, mientras que los lenguajes regulares están formados, potencialmente, por un número infinito de cadenas).

2.3. Autómatas finitos y expresiones regulares

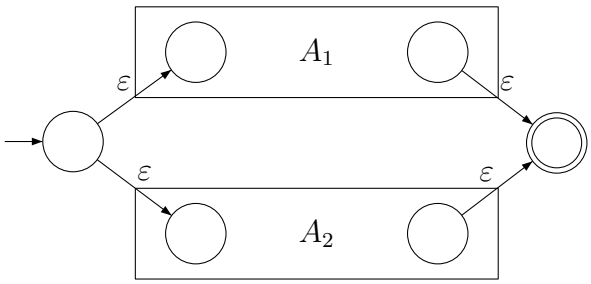
Para concluir el Capítulo, veamos en esta última Sección que las expresiones regulares describen, al igual que los autómatas o las gramáticas regulares, a los lenguajes regulares. Para ello, veamos en primer lugar que el lenguaje descrito por una expresión regular es aceptado por un autómata finito.

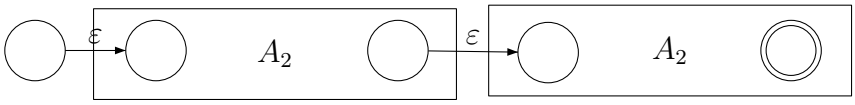
Teorema 29 (Equivalencia entre autómatas finitos y e.r.) *Sea Σ un alfabeto finito y E una expresión regular sobre dicho alfabeto. Entonces, existe un autómata finito (no determinista con transiciones vaciás) $A = (K, \Sigma, \delta, q_0, F)$ tal que $L(A) = L(E)$.*

Demostración .– La demostración es por inducción estructural en la expresión regular; como subproducto, se describe un algoritmo que transforma expresiónes regulares en autómatas finitos (no deterministas con transiciones vaciás para más precisos)
(Caso Base) Supongamos, de entrada, que la exprexión regular es alguna de las siguientes: $\emptyset$, ε o σ (con $\sigma \in \Sigma$. Consideramos los siguientes autómatas:



Está claro que el primer autómata acepta el lenguaje de la e.r. $\emptyset$, el segundo autómata acepta el lenguaje de la e.r. ε) y, por último, el tercer autómata acepta el lenguaje la expresión regular σ.
(Paso inductivo).- Supongamos ahora que $E = E_1E_2$ ó $E = E_1 + E_2$, donde E_1 y E_2 son dos exprexiones regulares con A_1 y A_2 como autómatas finitos asociados (i.e. se satisface que $L(E_1) = L(A_1)$ y $L(E_2) = L(A_2)$). Nótese que la existencia de dichos autómatas se sigue por hipótesis de inducción. Consideramos, dependiendo del caso, los siguientes autómatas:





Queda claro que el lenguaje del primer autómata es, precisamente, $L(E_1) \cup L(E_2)$ (i.e. el lenguaje de la expresión $E_1 + E_2$), mientras que el lenguaje del segundo es $L(E_1) \cup L(E_2)$ (i.e. el lenguaje de E_1E_2).
Gracias a la construcción anterior, podemos construir inductivamente, dada una expresión cualquiera E, un autómata finito A tal que $L(A) = L(E)$. Con lo que queda probado el resultado. ■

TODO 3 ***Ejemplo 30 (Conversión de AFD a ER)***

Veamos por último que todo lenguaje regular está descrito por una expresión regular; más concretamente, veamos que todo autómata acepta un lenguaje descrito por una expresión regular.

Teorema 31 *Sea $A = (K, \Sigma, \delta, q_0, F)$ un autómata finito sobre un alfabeto Σ. Entonces, existe una expresión regular sobre el mismo alfabeto Σ, digamos E, tal que $L(E)$=$L(A)$.*

Demostración .– En lo que sigue, y para simplificar, supondremos que el autómata A es un autómata finito determinista. Si no es el caso, basta convertirlo en determinista mediante los algoritmos vistos con anterioridad. Además, supondremos que el conjunto de estados es de la forma $\{1, \ldots, n\}$ (basta renombrarlos si no es así); en particular, asumimos que el número de estados de A es n.
Construimos una familia de expresiones regulares R_{ij}^k, con $1 \leq k \leq n$ y $1 \leq i, j \leq k$, cuyo interpretación es la siguiente: R_{ij}^k es la expresión regular que describe las cadenas sobre Σ que hacen que el autómata pase del estado i al estado j atravesando, únicamente, nodos etiquetados con naturales menores o iguales a k. La construcción de dichas expresiones regulares es inductiva en k y se detalla a continuación.
En primer lugar, supongamos que $k = 0$ (caso base). Entonces, R_{ij}^0 denota el conjunto de las cadenas que hacen transicionar al autómata A del estado i al estado j sin pasar por estado alguno (todos los estados están etiquetados con un natural mayor que cero). Por lo tanto, caben dos posibilidades:

- existe una transición del nodo i al nodo j o, equivalentemente, existe un camino (arco) en el diagrama de transición de A que va de i a j; suponiendo que dicho camino está etiquetado con los símbolos $\sigma_1, \ldots, \sigma_l$, definimos $R_{ij}^0 = \sigma_1 + \ldots + \sigma_l$,
- no existe una tal transición; en este caso, definimos $R_{ij}^0 = \emptyset$.

Supongamos construidas las expresiones regulares R_{ij}^k para $1 \leq k \leq n$. Construyamos la expresiones regulares R_{ij}^{n+1}, con $1 \leq i, j \leq n+1$. Dados i, j, con $1 \leq i, j \leq n+1$, supongamos que existe una caden que hace transicionar al autómata del estado i al j atravesando estados etiquetados con naturales menores o iguales que $n+1$. Distinguimos los siguientes casos:

- si existe la cadena en cuestión no hace transicionar al autómata A por el estado $n+1$ (la cadena da un camino del nodo i al nodo j que no

atraviesa nodos etiquetados con naturales mayores que n en el diagrama de transiciones), dicha cadena está comtemplada en la expresión regular R^n_{ij},

- si no es el caso, esto es, la cadena hace transicionar al autómata por el estado n, partimos dicha cadena en fragmentos de la siguiente manera:

  - el primer fragmento hace transicionar al autómata del estado i al $n+1$ sin pasar por $n+1$ como estado intermedio,
  - el último fragmento hace transicionar al autómata del estado $n+1$ al estado j sin pasar por $n+1$ como estado intermedio, y
  - los fragmentos intermedios son tales que hacen pasar al autómata del estado $n+1$ al estado $n+1$ sin que éste sea uno de los intermedios.

  Queda claro que el primer fragmento está descrito mediante la e.r. $R^n_{i(n+1)}$, el último está descrito por la e.r. $R^n_{(n+1)j}$ y los fragmentos intermediarios están descritos por la e.r. $R^n_{(n+1)(n+1}$. En particular, dicha cadena pertenece al lenguaje de la e.r. $R^n_{i(n+1)}(R^n_{(n+1)(n+1})^*R^n_{(n+1)j}$.

Por lo tanto y en cualquier caso, dicha cadena queda descrita por la expresión regular

$$R^n_{ij} + R^n_{i(n+1)}(R^n_{(n+1)(n+1})^*R^n_{(n+1)j}.$$

De la discusión anterior se sigue que la expresión regular que da el lenguaje del autómata es la dada por la suma de las expresiones regulares de la forma R^n_{1k} donde 1 suponemos es el estado inicial y k es un estado final. ∎

TODO 4 ***Ejemplo 32 (Conversión de ...)***

Capítulo 3

Propiedades de los lenguajes regulares

Tal y como indica el título, a lo largo del presente Capítulo estudiaremos diversas propiedades de los lenguajes regulares.

De entrada, veremos que no todos los lenguajes son regulares (viendo para ello el Lema de Bombeo). Hecho esto, pasaremos a estudiar las propiedades de clausura y de decisión de los mismos.

3.1. Lenguajes no regulares

A lo largo de esta Sección demostraremos la existencia de lenguajes no regulares. Para ello, demostraremos en primer lugar el *Lema de Bombeo*. Dicho Lema se demuestra gracias al conocido *Principio de las cajas*: si en n cajas se encuentran distribuidos $n+1$ objetos, existe una caja en la que hay, al menos, dos objetos de la colección.

Antes de entrar en los detalles teóricos veamos una ejemplo sencillo de lenguaje no regular.

Ejemplo 33 *Considérese el lenguaje* $L := \{0^n 1^n : n \in \mathbb{N}\}$ *definido sobre el alfabeto* $\{0, 1\}$. *Intuitivamente se puede llegar a la conclusión de que dicho lenguaje no es regular puesto que un autómata finito que lo reconozca debe ser capaz de contar el número de ceros para compararlo con el número de unos. Sin embargo, dado que los autómatas tienen una capacidad de almacenamiento limitada por el número de estado, no pueden almacenar números arbitrariamente grandes. Por lo tanto, no puede existir un tal autómata, i.e.* L *no es regular*

Una prueba formal de la no regularidad del lenguaje del Ejemplo anterior requiere la utilización del Lema de Bombeo.

Lema 34 (Lema de Bombeo) *Sea L un lenguaje regular. Entonces, existe una constante n (dependiente de L) tal que para toda cadena $\omega \in \Sigma^*$ de longitud mayor o igual que n existen cadenas $\omega_1, \omega_2, \omega_3 \in \Sigma^*$ tales que:*

- $\omega_1 \neq \varepsilon$,
- $|\omega_1\omega_2| \leq n$, *y*
- *para todo $k \in \mathbb{N}$, la cadena $\omega_1\omega_2^k\omega_3$ pertenece a L.*

Demostración .– Supongamos dado un lenguaje regular L y consideremos un autómata finito, digamos $A = (Q, \Sigma, \delta, q_0, F)$, i.e. $L = L(A)$. Supongamos además que el conjunto de estados consta de $n \in \mathbb{N}$ elementos, i.e. $\#Q = n$. Dada una cadena $\omega = \sigma_1 \cdots \sigma_m \in \Sigma^*$, con $m \geq n$, definimos p_i, para $0 \leq i \leq m$, como el estado en el que se encuentra el autómata tras procesar la cadena $\sigma_1 \cdots \sigma_i$, i.e. $p_i = \hat{\delta}(q_0, \sigma_1 \cdots \sigma_i)$[1].
Obviamente, existen $0 \leq j < k \leq n$ tales que $p_j = p_k$. Definiendo $\omega_1 := \sigma_1 \cdots \sigma_j$, $\omega_2 := \sigma_{j+1} \cdots \sigma_k$, y $\omega_3 := \sigma_{k+1} \cdots \sigma_m$, se sigue el resultado. ∎

Teniendo en cuenta lo anterior, estamos en disposición de demostrar la no regularidad del lenguaje del Ejemplo 33.

Ejemplo 35 *Supongamos, razonando por reducción al absurdo, que el lenguaje del Ejemplo 33 es regular. Si así fuera, por el Lema de Bombeo, existiría un natural $n \in \mathbb{N}$ tal que para toda cadena de longitud como poco n, digamos $\omega \in \Sigma^*$ existen cadenas $\omega_1, \omega_2, \omega_3 \in \Sigma^*$ tales que*

1. $\omega_1 \neq \varepsilon$,
2. $|\omega_1\omega_2| \leq n$, *y*
3. *para todo $k \in \mathbb{N}$, la cadena $\omega_1\omega_2^k\omega_3$ pertenece a L.*

En particular, consideramos la cadena de L definida por $\omega = 0^n1^n$. Por lo anterior, existen $\omega_1, \omega_2, \omega_3$ verificando 1, 2 y 3. Tomando $k = 0$, se sigue que $\omega_1\omega_3$ pertenece a L, lo cual es absurdo puesto que su número de ceros es forzosamente menor que él de unos (ω_2 tiene longitud mayor o igual a uno, i.e. consta de como poco un cero).

En resumen, hemos llegado a una contradicción en nuestro razonamiento. Dicha contradicción viene de suponer que L es un lenguaje regular.

TODO 5 *Veamos otro ejemplo sencillo de lenguaje no regular.*

Ejemplo 36 *Consideremos el lenguaje de las palabras cuya longitud es un númeor primo.*

[1] Nótese que $p_0 = q_0$.

3.2. Propiedades de clausura

A lo largo de esta Sección estudiaremos lo que se ha dado en llamar las propiedades de clausura de los lenguajes regulares, i.e. su comportamiento frente a uniones, intersecciones, diferencias, transformaciones, clausuras, concatenaciones,...

Comenzaremos tratando las propiedades de clausura de operaciones típicamente conjuntistas.

Teorema 37 *Las siguientes afirmaciones son ciertas.*

1. *La unión de dos lenguajes regulares es regular.*
2. *El complementario de un lenguaje regular es regular.*
3. *La intersección de dos lenguajes reuglares es regular.*
4. *La diferencia de dos lenguajes regulares es regular.*

Demostración .– 1. Sean $L, L' \subseteq \Sigma^*$ dos lenguajes regulares y sean E_L y $E_{L'}$ las expresiones regulares que dan dichos lenguajes. La expresión regular $E_L + E_{L'}$) determina, precisamente, el lenguaje $L \cup L'$ y, por lo tanto, dicho lenguaje es regular.
2. Sea L un lenguaje regular determinado por un autómata finito determinista A. Construimos un autómata A' igual a A con la salvedad de que sus estados aceptadores son los estados de rechazo de A. El autómata A', así construido, acepta cualquier cadena que A rechaze, i.e. determina el lenguaje L^c. Por lo tanto, el complementario de L es un lenguaje regular.
3. Dados dos lenguaje regulares $L, L' \subset \Sigma^*$, se tiene, gracias a las Leyes de Morgan:

$$L \cap L' = (L^c \cup L'^c)^c \, .$$

Puesto que, tal y como se ha demostrado en 1. y 2., la unión y el complementario de lenguajes regulares es regular, se sigue que la intersección es regular.
4. Sean $L, L' \subseteq \Sigma^*$ dos lenguajes regulares. Puesto que se verifica:

$$L \setminus L' = L \cap L'^c,$$

se sigue, gracias a 1. y 3., que la diferencia de lenguajes regulares es regular. ■

Veamos a continuación las propiedades de clausura de los lenguajes regulares frente a diversas transformaciones comenzando por la reflexión. Dicha transformación se define, en el caso de cadenas de la siguiente manera.

Definición 38 (Reflexión de una cadena) *Dada una cadena cualquiera $\omega = \omega_1 \cdots \omega_n \in \Sigma^*$ sobre el alfabeto Σ, se define la reflexión de la cadena ω, denotada por ω^R, como*

$$\omega^R := \omega_n \cdots \omega_1.$$

En cuanto al caso de lenguajes, la definición precisa es como sigue.

Definición 39 (Reflexión de un lenguaje) *Dado un lenguaje $L \subseteq \Sigma^*$, se define la reflesión del lenguaje L, denotada por L^R, como*

$$L^R := \{\omega^R : \omega \in L\}.$$

Teniendo en cuenta las definiciones anteriores se tiene el siguiente resultado.

Teorema 40 *Sea $L \subseteq \Sigma$ un lenguaje regular. Entonces, L^R es un lenguaje regular.*

Demostración .– Para demostrar el enunciado haremos uso de la relación existente entre lenguajes regulares y expresiones regulares.
Supongamos en primer lugar que el lenguaje regular L es descrito por una expresión regular E. Procediendo por inducción estructural, distinguimos los siguientes casos:

- (Caso Base) Si $E = \emptyset$, $E = \varepsilon$ o $E = \sigma$, para algún $\sigma \in \Sigma$, definiendo E^R como E se tiene

$$L(E^R) = L(E)^R = L.$$

- (Paso inductivo) Consideremos las restantes alternativas:

  - Supongamos que $E = E_1 + E_2$, para ciertas expresiones regulares E_1 y E_2. Definiendo $E^R = E_1^R + E_2^R$ tenemos que

$$\begin{aligned} L(E^R) &= L(E_1^R + E_2^R) = L(E_1^R) \cup L(E_2^R) \\ &= L(E_1)^R \cup L(E_2)^R = L(E_1 + E_2)^R = L(E). \end{aligned}$$

  - Con respecto al caso $E = E_1E_2$, definiendo $E^R = E_2^R E_1^R$ tenemos

$$\begin{aligned} L(E^R) &= L(E_2^2 E_1^R) = L(E_2^R)L(E_1^R) = L(E_2)^R L(E_1)^R \\ &= (L(E_1)L(E_2))^R = L(E_1E_2)^R = L(E)^R. \end{aligned}$$

  - Finalmente, el caso $E = E_1^*$ se sigue del anterior.

■

La demostración del Teorema anterior es constructiva: da un algoritmo recursivo para la construcción de la expresión regular asociada a la reflesión de un lenguaje tomando como entrada una expresión regular para el mismo. Veamos un ejemplo sencillo que ilustre tal extremo.

Ejemplo 41 *Consideremos la expresión regular* $E = (0+1)^*01$ *que determina el lenguaje de las cadenas de ceros y unos terminadas en* 01*. Dicha expresión es el resultado de la concatenación de tres expresiones regulares:* $(0+1)^*$*,* 0 *y* 1*. Por lo tanto, su reflexión coincide con la concatenación, en orden inverso, de la reflexión de dichas expresiones regulares, a saber:* $10(0+1)^*$*.*

Otra transformación usual, dentro del presente contexto, es la del cambio en la codificación de las cadenas. Para formalizar dicho cambio introducimos el concepto de homomorfismo.

Definición 42 (Homomorfismo) *Dados dos alfabetos finitos* Σ *y* Γ*, llamaremos homomorfismo a toda aplicación* $h : \Sigma \to \Gamma^*$*. Toda aplicación como la anterior induce, de manera natural una aplicación* $h : \Sigma^* \to \Gamma^*$ *definida mediante*

$$h(\sigma_1 \cdots \sigma_n) := h(\sigma_1) \cdots h(\sigma_n).$$

Dado un lenguaje cualquiera $L \subseteq \Sigma$*, llamaremos homomorfismo de* L *(con respecto a* h*) al lenguaje* $h(L)$ *definido por*

$$h(L) = \{\omega \in \Gamma : \exists \omega' \in L t.q.\ h(\omega') = \omega\}.$$

Una vez formalizado el concepto, veamos cómo se comporta el concepto de regularidad frente al mismo.

Teorema 43 *Sea* $L \subset \Sigma^*$ *un lenguaje regular y* $h : \Sigma \to \Gamma$ *un homomorfismo entre los alfabetos (finitos)* Σ *y Gamma. Entonces, el homomorfismo del lenguaje* L *(con respecto a* h*) es un lenguaje regular.*

Demostración .– Supongamos que el lenguaje L viene dado mediante una expresión regular sobre el alfabeto Σ, digamos E. Construyamos, por inducción estructural, una nueva expresión, digamos E^h, que determine $h(L)$.

- (Caso Base) Supongamos de entrada que $E = \emptyset$, $E = \varepsilon$ ó $E = \sigma$ (con $\sigma \in \Sigma$). Entonces, definimos E^h mediante $E^h := \emptyset$, $E^h := \varepsilon$ ó $E^h := h(\sigma)$ de pendiendo del caso. Está claro que, en este caso, $h(L(E)) = L(E^h)$.

- (Paso Inductivo) Distinguimos los siguientes casos:
  - si $E = E_1 + E_2$, definimos $E^h := E_1^h + E_2^h$.
  - si $E = E_1 E_2$, definimos $E^h := E_1^h E_2^h$ y, finalmente,
  - si $E = E_1^*$, definimos $E^h := (E^h)^*$. Resulta fácil de ver que en los tres casos anteriores se tiene que $h(L(E)) = L(E^h)$.

En resumen, el homomorfismo de un lenguaje regular es regular. ∎

Otra transformación usual es la del homomorfismo inverso.

Definición 44 (Homomorfismo inverso) *Sean Σ y Γ dos alfabetos finitos, $h : \Sigma \to \Gamma$ un homomorfismo y $L \subset \Gamma^*$ un lenguaje regular. Se define el homomorfismo inverso de L (con respecto a h), denotado $h^{-1}(L)$, como:*

$$h^{-1}(L) := \{\omega \in \Sigma^* : h(\omega) \in L\}.$$

Al igual que en casos anteriores, la regularidad se conserva. Más concretamente,

Teorema 45 *Sean Σ y Γ dos alfabetos finitos, $h : \Sigma \to \Gamma$ un homomorfismo y $L \subset \Gamma^*$ un lenguaje regular. El homomorfismo inverso de L es un lenguaje regular.*

Demostración .– Supongamos dado un autómata $A = (K, \Gamma, \delta, q_0, F)$ que determina el lenguaje L, i.e. $L(A) = L$. Construyamos un autómata $A' = (K, \Sigma, \delta', q_0, F)$ (comparte con A el conjunto de estados, el estado inicial y los estados aceptadores) definiendo la función de transición de A' mediante

$$\delta'(q, \sigma) := \hat{\delta}(q, h(\sigma)),$$

donde $\hat{\delta}$ denota, como es usual, la función de transición extendida de A.
Así definido A', queda claro que el lenguaje que acepta es precisamente $h^{-1}(L)$. ∎

3.3. Propiedades de decisión

A lo largo de esta sección estudiaremos algunas propiedades decidibles de los lenguajes regulares, i.e. cuestiones que se pueden resolver de manera algorítmica.

Típicamente, un lenguaje regular está constituido por un número infinito de cadenas por lo que, dado nuestro interes en tratar cuestiones de manera algorítmica, debemos representarlo de manera más suscinta que la enumeración de todas y cada una de sus cadenas. Para ello, podemos suponer que el lenguaje está descrito mediante un autómata finito o una expresión regular. Esta independencia de la representación se sigue del siguiente teorema:

Teorema 46 *Existen algoritmos que llevan a cabo las siguientes tareas:*

1. *Dada una expresión regular E, construir un autómata finito A (determinista, no determinista o no determinista con transiciones vacías) tal que $L(A) = L(E)$.*
2. *Dado un autómata finito A (determinista, no determinista o no determinista con transiciones vacías), construir una expresión regular E tal que $L(E) = L(A)$.*

Demostración .– La demostración del apartado 1. puede encontrarse en la del Teorema 29, mientras que la prueba de 2. puede verse en la demostración del Teorema 31. ∎

Una vez clarificado lo tocante a la representación de lenguajes regulares, el primer problema que uno se plantea es el siguiente:

Cuestión 47 *Dado un lenguaje regular (bien mediante un autómata finito, una expresión regular o una gramática regular), ¿es L no vacío?*

Si el lenguaje viene dado mediante un autómata finito, dicha pregunta se puede interpretar en términos de accesibilidad de grafos de la siguiente manera: ¿existe un camino del estado inicial a un estado aceptador? Así enunciada, la pregunta tiene fácil respuesta algorítmica:

- el estado inicial es accesible desde el estado inicial, y
- si el estado p es accesible desde el estado inicial y existe un arco de p a q, entonces el estado q es accesible desde el estado inicial.

Lo anterior resume la construcción inductiva del conjunto de estados accesibles desde el estado inicial, en el momento en que dicho conjunto contenga un estado aceptador, concluimos que el lenguaje no es vacío. Si dicho conjunto no se puede ampliar y no contiene ningún estado aceptador, concluimos que el lenguaje L es vacío.

En el caso de lenguajes dados mediante expresiones regulares podemos proceder, por inducción estructural, de la manera siguiente:

- (Caso Base) Si la expresión regular es $\emptyset$ define el lenguaje vacío; si es del tipo ε o σ ($\sigma \in \Sigma$), entonces el lenguaje no define vacío.
- (Paso Inductivo) Si la expresión regular no es de los tipos anteriores, distinguimos los siguientes casos:
  - Si es de la forma $E_1 + E_2$, el lenguaje es vacío si y sólo si los lenguajes de E_1 y E_2 lo son,
  - Si es del tipo E_1E_2, el lenguaje es vacío si y sólo si el lenguaje de E_1 o el de E_2 lo son.
  - Finalmente, si la expresión regular es del tipo E^*, el lenguaje no es vacío.

Si el lenguaje viene dado por una gramática regular, basta transformarla en un autómata, gracias al algoritmo visto en el Capítulo anterior, y determinar si éste decide el lenguaje vacío o no.

En cualquier caso, tenemos el siguiente teorema:

Teorema 48 *Existe un algoritmo que dado un lenguaje regular L, descrito mediante un autómata finito o una expresión regular, decide si es o no vacío.*

Otra cuestión importante es la siguiente:

Cuestión 49 *Dado un lenguaje regular (descrito mediante un autómata finito, una expresión regular o una gramática regular) y una cadena sobre el mismo alfabeto, ¿pertenece la cadena al lenguaje?*

Puesto que las tres representaciones son equivalentes (existen algoritmos que pasan de unas a otras), supongamos simplemente que el lenguaje viene dado por un autómata finito determinista. Para concluir si la cadena pertenece o no al lenguaje, bastará con simular el autómata. En resumen, tenemos el teorema:

Teorema 50 *Existe un algoritmo que dado un lenguaje regular L y una cadena sobre el mismo alfabeto decide si la cadena pertenece o no al lenguaje.*

3.4. Equivalencia y minimización

Comenzamos esta Sección estudiando suscintamente la noción de relación de equivalencia, de utilidad en lo que sigue.

Definición 51 (Relación de equivalencia) *Sea X un conjunto. Una relación $\sim \subseteq X \times X$ se dice de equivalencia, si satisface las siguientes propiedades:*

- *[Reflexiva] para todo $x \in X$, $x \sim x$,*
- *[Simétrica] para todos $x, y \in X$, $x \sim y$ implica $y \sim x$, y*
- *[Transitiva] para todos $x, y, z \in X$, $x \sim y$ y $y \sim z$ implica $x \sim z$.*

Dada una relación $\sim \subseteq X \times X$ y un elemento $x \in X$, se define la clase de x, denotada mediante $[x]_\sim$, como el conjunto de todos los elementos de X relacionados con x, i.e.

$$[x] := \{y \in X : x \sim y\}.$$

Además, dados dos elementos $x, y \in X$ no relacionados mediante R, se verifica:

$$[x] \cap [y] = \emptyset.$$

La demostración de este hecho se sigue la propiedad transitiva. Veamos su demostración en detalle. Por reducción al absurdo, sean $x, y \in X$ dos elementos no relacionados tales que

$$[x] \cap [y] \neq \emptyset.$$

y sea $z \in [x] \cap [y]$. Puesto que $x \sim z$ y $z \sim y$ se sigue $x \sim y$, lo cúal es absurdo. Por lo que se tiene la afirmación.

Gracias a este hecho, podemos dar una partición del conjunto X en clases de equivalencia.

En lo que sigue, emplearemos el concepto de relación de equivalencia en la minimización y equivalencia de autómatas. Para ello, introducimos la noción de equivalencia de estados.

Definición 52 (Estados equivalentes) *Sea $A = (Q, \sigma, \delta, q_0, F)$ un autómata finito determinista y sean $p, q \in Q$ dos estados del mismo. Diremos que los estados p y q son equivalentes, denotado por $p \sim q$, si para toda cadena $w \in \Sigma^*$ se verifica que $\hat{\delta}(p, w)$ es de aceptación si y sólo si $\hat{\delta}(q, w)$ es de aceptación. En caso contrario, diremos que los estados son distinguibles.*

Teorema 53 *La relación* estados equivalentes *es una relación de equivalencia.*

Demostración .– Las tres propiedades se siguen directamente de la definición, omitiremos su prueba. ∎

Además de ser una relación de equivalencia en el conjunto de estados del autómata, se pueden determinar las clases de equivalencia.

Teorema 54 *Existe un algoritmo que tomando como entrada un autómata finito determinista, calcula los pares de estados distinguibles y por ende, los pares de estados equivalentes.*

Demostración .– El algoritmo que lleva a cabo la tarea anunciada procede como sigue:

- (Caso Base) Si $p \in Q$ es un estado de aceptación y $q \in A$ no lo es, se marca el par (p, q) como par de estados distinguibles. Dichos estado se distinguen mediante la cadena ε.

- (Paso Inductivo) Sean $p, q \in Q$ dos estados tales que para algún $\sigma \in \Sigma$ se tiene $\delta(p, \sigma) = s$ y $\delta(q, \sigma) = r$. Si el par (r, s) esta marcado como distinguible, se marca el par (p, q) como distinguible. Obviamente, una cadena que distingue p y q es aw, donde $w \in \Sigma^*$ es una cadena que distingue a r y s.

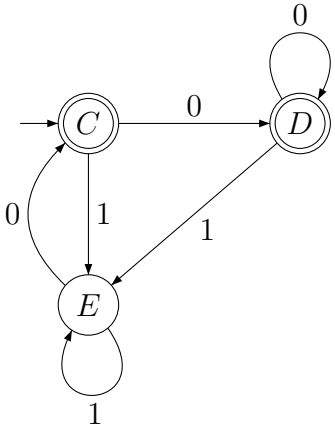
El algoritmo anterior marca todos los pares de estados distinguibles. Para demostrarlo, supongamos que no es así, i.e. que existe un par de estado $p, q \in Q$ distinguibles que no han sido marcados por el algoritmo.

Los estados p y q no pueden ser distinguidos mediante la cadena ε, puesto que si así fuera hubieran sido marcados en la primera etapa del algoritmo (uno sería aceptador mientras que el otro no).

Sea $w = w_1 \dots w_n \in \Sigma^+$ la cadena más corta que distingue a p y q, los estados $\delta(p, w_1)$ y $\delta(q, w_1)$ son distinguidos por la cadena $w_2 \dots w_n$. Continuando con la argumentación, llegamos a la conclusión de que unos ciertos estados $r \in Q$ y $s \in Q$ son distinguibles por la cadena vacía y por tanto conforman un par marcado en la primera etapa del algoritmo. Por lo tanto, el par conformado por p y q hubiera sido marcado durante la ejecución de la segunda etapa. ■

Veamos un ejemplo concreto.

Ejemplo 55 *Sea A el autómata finito dado por el siguiente diagrama de transición:*



En este caso, el algoritmo comienza marcando lo pares formados por estados aceptadores y no aceptadores, i.e.:

$$\{(C, E), (D, E)\}.$$

Por concisión eliminamos las repeticiones del tipo (E, D) o (E, C). Tras esta primera fase, el algoritmo pasa a la fase inductiva. Sin embargo, en esta fase no marca nuevos pares puesto que el par (D, C) es indistinguible.

Una primera aplicación del algoritmo detallado en el Teorema 56 resulta ser la existencia de un algoritmo que decide la igualdad de dos lenguajes dados mediante autómatas finitos.

Teorema 56 *Existe un algoritmo que decide si dos autómatas finitos aceptan el mismo lenguaje.*

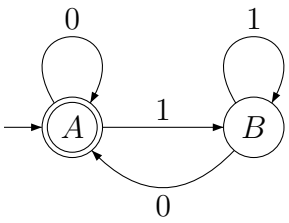
Demostración .– Supongamos dados dos autómatas finitos

$$A = (Q_A, \Sigma_A, \delta_A, q_0^A, F_A) \text{ y } B = (Q_B, \Sigma_B, \delta_B, q_0^B, F_B)$$

tales que sus conjuntos de estados sean disjuntos. Consideramos un nuevo autómata cuyo conjunto de estados viene dado por la unión de los conjuntos Q_A y Q_B. El resto de los elementos del nuevo autómata se definen de manera acorde a esta elección.
Cabe notar que el autómata así definido, tiene dos estados iniciales; sin embargo esto no afecta a lo que sigue.
El algoritmo emplea el algoritmo de marcado del Teorema 56 para detectar si los estados iniciales son equivalentes o no. Si así es, ambos autómatas definen el mismo lenguaje, en caso contrario no. ■

El Teorema anterior puede ser demostrado haciendo uso de los algoritmos vistos en la Sección 3.3. Para ello, basta observar que dos lenguajes L_1 y L_2 coinciden si y sólo si $L_1 \subseteq L_2$ y $L_2 \subseteq L_1$ o equivalentemente, si $L_1 \setminus l_2 = \emptyset$ y $L_2 \setminus L_1 = \emptyset$. Por lo tanto, se reduce el problema de la igualdad de lenguajes al de vacuidad.

Ejemplo 57 *Consideremos el autómata del Ejemplo 55 y el dado por el diagrama de transición siguiente:*



Empleando el algoritmo de marcado del Teorema 54 se tiene que son distinguibles los pares de estados (y sus simétricos):

$$\{(B,A),(E,A),(C,B),(D,B),(E,C),(E,D)\}$$

Por lo tanto, el par (A,C) es un par de estados equivalentes, por lo que ambos autómatas definen el mismo lenguaje.

Otra aplicación del algoritmo de marcado es la minimización del número de estados de un autómata.

Teorema 58 *Existe un algoritmo que tomando como entrada un autómata finito determinista, produce un autómata equivalente con el menor número posible de estados.*

Demostración .– Supongamos dado un autómata $A = (Q, \Sigma, \delta, q_0, F)$. El algoritmo elimina de entrada aquellos estados que no son accesibles desde el estado inicial. Para ello puede emplear el algoritmo visto en la Sección 3.3. Después, construye un nuevo autómata $B = (Q', \Sigma, \delta', q_0', F')$ de la siguiente manera:

- Q' es el conjunto de clases de equivalencia de Q con respecto a $\sim$ (equivalencia de estados),
- q_0' es la clase de equivalencia de q_0, i.e. $q_0' := [q_0]_\sim$,

- F' es el conjunto de clases de equivalencia de los elementos de F, y
- δ' se define de la siguiente manera: dado $\sigma \in \Sigma$ y $q \in Q'$, se define $\delta'(q, \sigma) = [\delta(q, \sigma)]_\sim$. La función de transición esta bien definida puesto que, si $p \in [q]_\sim$ y $[\delta(q, \sigma)]_\sim \neq [\delta(p, \sigma)]_\sim$, los estados p y q serían distinguibles.

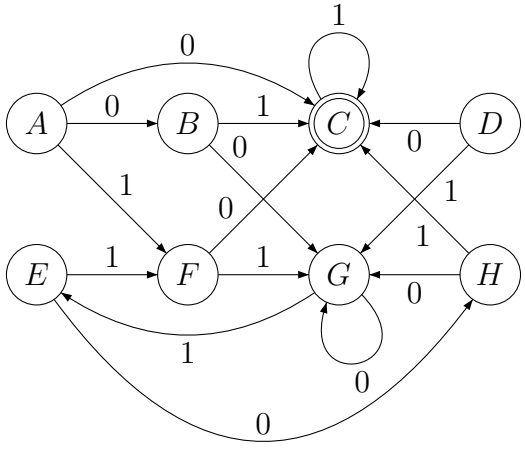
El algoritmo, para construir un tal autómata, hace uso del algoritmo de marcado del Teorema 56 con el fin de determinar las clases de equivalencia. Veamos que el autómata B así construido tiene el menor número de estados posibles. Por reducción al absurdo supongamos que no es así y sea C un autómata con un número menor de estados. Como definen el mismo lenguaje, i.e. son autómatas aquivalentes, los estados iniciales son equivalentes. Como ninguno de los dos autómatas posee estados aislados del inicial, cada estado de B va a ser equivalente a alguno de C. Puesto que C tiene menos estados que B, hay dos estados de B equivalentes entre si, lo cual resulta absurdo. ■

Un sencillo ejemplo de la construcción dada en el Teorema anterior es el siguiente.

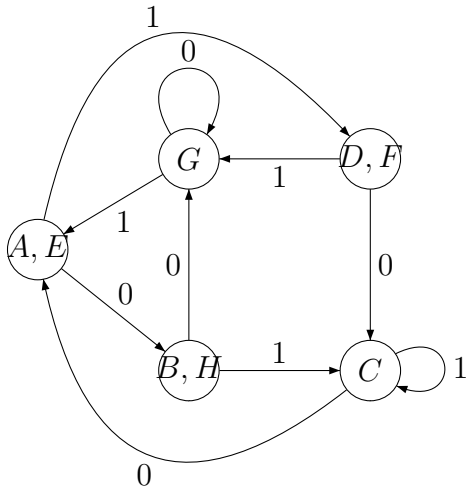
Ejemplo 59 *Es fácil ver que el autómata del Ejemplo 57 se obtiene a partir del autómata del Ejemplo 55 aplicando el algoritmo del Teorema anterior.*

Otro ejemplo, esta más complicado es el siguiente:

Ejemplo 60 *Considerese el siguiente autómata:*



El mínimo autómata que acepta el lenguaje del anterior es:


1
0
G
D, F
1
1
A, E
0
0
0
B, H
1
C
1
0

Capítulo 4

Autómatas a pila

Hasta el momento hemos tratado los autómatas finitos. Dichos autómatas pueden intrepertarse como máquinas abstractas que carecen de estructura de datos (sólo pueden almacenar información en un conjunto finito dado de estados). Hemos visto el potencial computacional de dichas máquinas y como pueden representarse los lenguajes que aceptan mediante gramáticas o expresiones regulares.

A lo largo del presente Capítulo, eliminaremos esta restricción y consideraremos autómatas dotados de la estructura de datos más sencilla que uno puede considerar: una pila. Dichos autómatas a pila serán también el centro de los dos siguientes temas.

4.1. Autómatas a pila

Tal y como se dijo con anterioridad, los autómatas a pila son autómatas finitos que cuentan con una pila. Por ello, en cada paso de computación de los mismos, son capaces de operar en la pila se acuerdo a las operaciones usuales de las mismas: obtener el contenido de la pila, apilar uno o más caracteres y desapilar un carácter. Su definición formal es como sigue.

Definición 61 (Autómata a pila) *Un autómata a pila es una tupla* $P = (K, \Sigma, \Gamma, \delta, q_0, Z, F)$ *donde:*

1. K *es un conjunto finito de estados,*
2. Σ *y* Γ *son dos alfabetos finitos llamados de entrada y de pila (respec.),*
3. $\delta : K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}(K) \times \Gamma^*$ *es una aplicación llamada de transición,*

4. $q_0 \in K$ *es el estado inicial,* Z *es el símbolo inicial de la pila, y*

5. F *es el conjunto de estados aceptadores.*

El lector puede observar las similitudes existentes entre la definición de autómata a pila y la de autómata finito.

A la hora de definir el funcionamiento de un autómata a pila, la aproximación más sencilla es la siguiente: primero se define el concepto de configuración (estado en el que se encuentra el autómata a pila en un momento concreto) y luego se define como él autómata pasa de una configuración a otra (paso de computación).

La definición formal de configuración es la siguiente:

Definición 62 (Configuración de un autómata a pila) *Dado un autómata a pila* $P = (K, \Sigma, \Gamma, \delta, q_0, Z, F)$*, llamaremos configuración del autómata* P *a toda tupla* $(q, w, \gamma) \in K \times \Sigma^* \times \Gamma^*$*. En particular, llamaremos configuración inicial para un cadena dada* $w \in \Sigma^*$ *a la tupla* (q_0, w, Z)*.*

Mientras que la de paso de computación es como sigue:

Definición 63 (Paso de computación en un autómata a pila) *Supongamos dados un autómata a pila* $P = (K, \Sigma, \Gamma, \delta, q_0, Z, F)$*, un símbolo del alfabeto de entrada* $\sigma \in \Sigma$ *y un símbolo del alfabeto de pila* $\rho \in \Gamma$ *tales que* $\delta(q, \sigma, \rho) = (Q, \alpha)$ *para ciertos* $Q \in \mathcal{P}(K)$ *y* $\alpha \in \Gamma^*$*. Para todo* $p \in Q$*,* $w \in \Sigma^*$ *y* $\gamma \in \Gamma^*$*, diremos que la configuración* $(q, \sigma w, \rho\gamma)$ *da lugar, en un paso de computación, a la configuración* $(p, w, \alpha\gamma)$ *y lo denotaremos por* $(q, \sigma w, \rho\gamma) \vdash_P (p, w, \alpha\gamma)$*. En general, emplearemos la notación* $\vdash_P^*$ *para indicar que una configuración se sigue de otra en un número finito de pasos (quizás nulo).*

Teniendo presentes las definiciones anteriores podemos definir el lenguaje aceptado por un autómata a pila de diversas maneras: por estado final o por pila vacía. La primera de ellas guarda estrecha relación con la manera en que se definen los lenguajes regulares, se considerán las cadenas que, tras procesarse, dejan al autómata en un estado aceptador.

Definición 64 (Lenguaje aceptado por estado final) *Dado un autómata a pila* $P = (K, \Sigma, \Gamma, \delta, q_0, Z, F)$*, se define el lenguaje aceptado por* P *por estado final como el conjunto de las cadenas* $w \in \Sigma^*$ *tales que partiendo de la configuración inicial para* w*, el autómata alcanza en un número finito de pasos una configuración aceptadora, i.e.*

$$L(P) = \{w \in \Sigma^* : (q_0, w, Z) \vdash_P^* (f, \varepsilon, \gamma) \in F \times \{\varepsilon\} \times \Gamma^*\}.$$

En la otra alternativa, aceptación por pila vacía, se exige que el autómata vacíe su pila (además de haber procesado su entrada) para aceptar una cadena.

Definición 65 (Lenguaje aceptado por pila vacía) *Dado un autómata a pila $P = (K, \Sigma, \Gamma, \delta, q_0, Z, F)$, se define el lenguaje aceptado por P por pila vacía como el conjunto de las cadenas $w \in \Sigma^*$ tales que partiendo de la configuración inicial para w, el autómata alcanza en un número finito de pasos una configuración con cadena y pila vacías, i.e.*

$$L(P) = \{w \in \Sigma^* : (q_0, w, Z) \vdash_P^* (f, \varepsilon, \varepsilon) \in K \times \{\varepsilon\} \times \{\varepsilon\}\}.$$

TODO 6 *Dar un ejemplo de autómata a pila que acepte un lenguaje por pila vacía y otro que acepte el mismo lenguaje por estado final.*

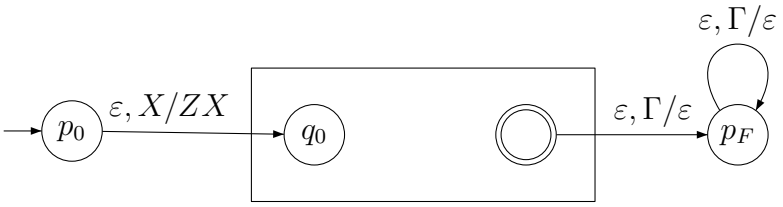
Aunque distintas, ambas definiciones nos llevan al mismo tipo de lenguajes, como muestra el siguiente Teorema; su interés reside la existencia de lenguajes para los cuales es más sencillo diseñar un autómata que acepte de una manera concreta.

Teorema 66 *Dado un lenguaje aceptado por un autómata P por estado final, existe un autómata P' que acepta L por pila vacía. Recíprocamente, si L es aceptado por un autómata por pila vacía, existe un autómata P' que acepta L por estado final.*

Demostración .– Supongamos dado un lenguaje $L \subset \Sigma^*$ aceptado por un autómnata a pila $P = (K, \Sigma, \Gamma, \delta, q_0, Z, F)$ por estado final. La construcción de un autómata P' que acepta L por pila vacía es como sigue: P' coincide con P con las siguientes salvedades:

- el símbolo inicial de pila viene dado por un nuevo símbolo X,
- el conjunto de estado coincide con el de P salvo por la inclusión de dos nuevos estados: un nuevo estado inicial p_0 y un nuevo estado denotado p_F encargado de vaciar la pila.
- la función de transición es igual que en el autómata a pila salvo por la inclusión de diversas transiciones: la primera está definida mediante $\delta(p_0, \varepsilon, X) = \{(q_0, ZX)\}$ y se encarag de apilar el símbolo inicial de pila del autómata P, mientras que las restantes son de la forma $\delta(f, \varepsilon, \gamma) = \{(p_F, \varepsilon)\}$, donde $f \in F$ y $\gamma \in \Gamma$.

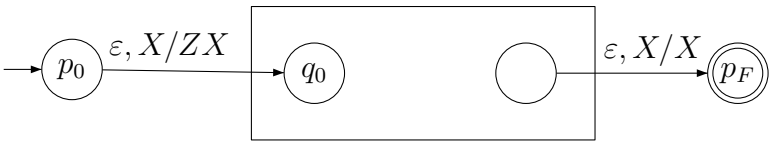
Gráficamente, representamos a continuación dicho autómata:



Recíprocamente, supongamos que el lenguaje L es aceptado por un autómata a pila por pila vacía. La construcción del autómata P' es como sigue: al igual que en el caso anterior, el nuevo autómata es esencialmente igual a P con las siguientes salvedades:

- el símbolo inicial de pila viene dado por un nuevo símbolo Z,
- se añaden dos nuevos estados p_0 (estado inicial) y p_F (estado final),
- se consideran, además de las transiciones de originales P, las siguientes: una transición inicial definida mediante $\delta(p_0, \varepsilon, X) = \{(q_0, ZX)\}$ y unas transiciones encargadas de aceptar definidas por $\delta(q, \varepsilon, Z) = \{(p_F, X)\}$.

En líneas generales, el diagrama de transiciones de P' tiene la pinta:



■

TODO 7 *Dar un ejemplo de la conversión.*

4.2. Autómatas a pila deterministas

Aparte de los autómatas a pila definidos en la Sección anterior, resultan interesantes los autómatas a pila deterministas. Hacemos hincapié en que aquí determinismo hace alusión a la inexistencia de diversas opciones para una misma situación. La definición formal es la siguiente:

Definición 67 (Autómata a pila determinista) *Dado un autómata a pila $P = (K, \Sigma, \Gamma, q_0, Z, F)$, diremos que es determinista si verifica:*

1. $\delta(q, \sigma, \gamma) = (Q, \gamma')$ *verifica* $\#Q \leq 1$ *para todo* $q \in K$, $\sigma \in \Sigma$ *y* $\gamma \in \Gamma$.
2. *Si* $\delta(q, \sigma, \gamma) = (Q, \gamma')$ *con* $Q \neq \emptyset$, *entonces* $\delta(q, \varepsilon, \gamma) = (\emptyset, \gamma')$.

Puesto que los autómatas a pila deterministas son, en particular, autómatas a pila, las definiciones de configuración y de paso de computación se aplican sin más. Por lo tanto, podemos definir el lenguaje aceptado por un autómata a pila por estado final o por pila vacía.

La relación de los autómatas a pila con los autómatas finitos viene expresada en el siguiente Teorema.

Teorema 68 *Dado un lenguaje regular $L \subseteq \Sigma^*$, exite un autómata a pila deterministas que acepta L por estado final.*

Demostración .– La demostración resulta sencillo y se deja como ejercicio al lector. ∎

Para finalizar, notemos que existen lenguajes aceptador por autómatas a pila que no son aceptados por autómatas a pila deterministas.

TODO 8 *Dar un ejemplo de lenguaje aceptado por un autómata a pila que no sea aceptado por un autómata a pila determinista.*

Capítulo 5

Gramáticas independientes del contexto

Al igual que los autómatas finitos tienen asociadas las gramáticas regulares, los autómatas a pila tienen asociadas las que se conocen como gramáticas independientes del contexto.

A lo largo del presente Capítulo introduciremos dichas gramáticas y detallaremos su relación con los autómatas a pila.

5.1. Gramáticas independientes del contexto

Comencemos con un ejemplo.

Ejemplo 69 (Palíndromos) *Dado un alfabeto Σ, con $\#\Sigma \geq 2$, un palíndromo sobre Σ es una cadena $w \in \Sigma$ tal que $w = w^R$. El lenguaje de los palíndromos, definido como el conjunto de los mismos, i.e.*

$$L_{pal} := \{w \in \Sigma^* : w = w^R\},$$

no es un leguaje regular. Para demostrar esta afirmación procederemos por reducción al absurdo. Asumamos que L_{pal} es regular. Por el Lema de Bombeo se tiene que existe un natural $n \in \mathbb{N}$ tal que para toda cadena $w \in L_{pal}$, con $|w| \geq n$, se tiene que existen cadenas $x, y, z \in \Sigma^*$ satisfaciendo las condiciones 1., 2. *y* 3. *del enunciado del Lema de Bombeo. En particular, dada la cadena $\alpha^n\beta\alpha^n \in L_{pal}$, donde $\alpha, \beta \in \Sigma$ son dos símbolos distintos, existen cadenas $x, y, z \in \Sigma^*$ satisfaciendo* 1., 2. *y* 3.*. Por lo que, en particular, se sigue que la cadena $xz \in L_{pal}$ o equivalentemente, $\alpha^m\beta\alpha^n \in L_{pal}$, con $m \in \mathbb{N}$ satisfaciendo $m < n$*[1]*. Lo cúal es absurdo puesto que $\alpha^m\beta\alpha^n$ no es un palíndromo.*

[1] Obsérvese que $y \neq \varepsilon$, por lo que $|y| \geq 1$, y que $|xy| \leq n$.

Del hecho de que L_{pal} no sea un lenguaje regular se sigue que no puede ser descrito mediante una expresión regular. Ahora bien, esto no implica que no pueda ser descrito de manera alguna.

Supongamos, en lo que resta de ejemplo, que $\Sigma = \{0, 1\}$ y consideremos las siguientes reglas de reescritura:

$$\begin{aligned} Q &\rightarrow \varepsilon, \\ Q &\rightarrow 0, \\ Q &\rightarrow 1, \\ Q &\rightarrow 0Q0, \text{ y} \\ Q &\rightarrow 1Q1. \end{aligned}$$

Tomando como punto de partida P y empleando las reglas de escritura anteriores, podemos generar cadenas de Σ^* de manera tal que aquellas que no contengan a P son palíndromos.

En resumen, L_{pal} puede describirse mediante un conjunto finito de reglas.

En lo que sigue precisaremos el concepto de gramática, de lo que el ejemplo anterior no es más que un caso particular.

Definición 70 (Gramática independiente del contexto) *Una gramática independiente del contexto es una tupla $G = (V, T, P, S)$ donde*

- *V es un conjunto finito de símbolos llamados variables o símbolos no terminales,*
- *T es un conjunto finito de símbolos llamados terminales,*
- *$P \subseteq V \times (V \cup T)^*$ es un conjunto finito de elementos, llamados producciones, de manera tal que si $(p, q) \in P$ se denotará mediante la expresión $p \rightarrow q$; p recibe el nombre de cabeza y q el de cuerpo, y*
- *$S \in V$ es el símbolo inicial.*

En el Ejemplo 69, empleabamos una gramática para construir las palabras del lenguaje de los palíndromos. Para ello, aplicabamos las producciones (o reglas de substitución) al símbolo inicial P y, de entre las cadenas generadas, aquellas que no contuvieran al símbolo P eran palíndromos.

Veamos, en primera instancia, como se define el proceso de substitución para gramáticas independientes del contexto cualesquiera.

Definición 71 (Derivación) *Dada una gramática independiente del contexto $G = (V, T, P, S)$, una cadena $\alpha A\beta$, con $\alpha, \beta \in (V \cup T)^*$ y $A \in V$, si existe una producción $A \to \gamma \in P$, diremos que $\alpha\gamma\beta$ deriva, en un paso, de $\alpha A\beta$; si así es, se denota mediante la expresión $\alpha A\beta \Rightarrow_G \alpha\gamma\beta$ (o equivalentemente $\alpha A\beta \Rightarrow \alpha\gamma\beta$ si esta claro la gramática empleada). En general, si una cadena $w \in (V \cup T)^*$ deriva en una o más pasos de $\theta \in (V \cup T)^*$, lo denotaremos mediante la expresión $\theta \Rightarrow_G^* w$ ($\theta \Rightarrow^* w$ si esta claro la gramática empleada).*

Observamos que en la definición de derivación no se especifica ni la variable a substituir ni la producción a emplear. Es, en cierto sentido, no determinista.

Si en una derivación, se substituye la variable que aparece más a la izquierda, se dice que es una derivación a la izquierda. Si por el contrario se substituye la variable más a la derecha, se dice que la derivación es a la derecha. En general, las cadenas derivables a partir del símbolo inicial pueden obtenerse bien mediante derivaciones cualesquiera, bien mediante derivaciones a izquierda o bien mediante derivaciones a derecha.

Las cadenas que aparecen como resultado de derivaciones del símbolo inicial suelen recibir el nombre de formas sentenciales.

Una vez formalizado el proceso de derivación (substitución), podemos definir el lenguaje descrito por una gramática independiente del contexto.

Definición 72 (Lenguaje independiente del contexto) *Dada una gramática independiente del contexto $G = (V, T, P, S)$, se define el lenguaje de G como el conjunto de cadenas de símbolos terminales derivables, en uno o más pasos, de S, i.e.*

$$L(G) := \{w \in T^* : S \Rightarrow_G^* w\}.$$

Para decidir si una cadena pertenece puede derivarse o no a partir de otra dada, basta determinar si existe una sucesión de derivaciones que comenzando con la última lleven a la primera. Este proceso se conoce con el nombre de inferencia recursiva.

Ejemplo 69 (Palíndromos, continuación) *Retomando el ejemplo de los palíndromos, y considerando la gramática $G = (V, T, P, S)$, donde $V := \{Q\}$, $T := \{0, 1\}$, $S := Q$ y P es el conjunto de las producciones allí descritas, tenemos por ejemplo la siguiente secuencia de derivaciones:*

$$P \Rightarrow_G 1P1 \Rightarrow_G 10P01 \Rightarrow_G 10101.$$

Puesto que 10101 *está compuesta exclusivamente por carácteres terminales, pertenece al lenguaje de los palíndromos.*

5.2. Árboles de derivación

Una secuencia de derivaciones puede representarse mediante un árbol. Formalmente.

Definición 73 (Árbol de derivación) *Dada una gramática independiente del contexto $G = (V, T, P, S)$, se define un árbol de derivación para G como uno que verifica las siguientes condiciones:*

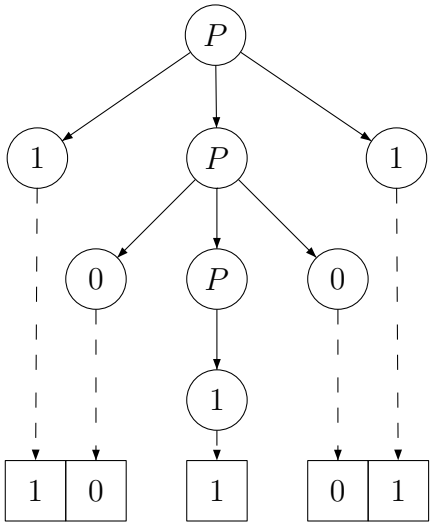
- *cada nodo interior está etiquetado con una variable de V,*
- *cada hoja está etiquetada con una variable, un carácter terminal o ε,*
- *si un nodo está etiquetado con una variable A y sus hijos están etiquetados con $X_1, \ldots, X_n$ (respetando el orden de aparición), entonces $A \to X_1 \cdots X_n$ es una producción de P.*

El resultado de un árbol de derivación es la concatenación de las etiquetas de las hojas de izquierda a derecha. Veamos un ejemplo tanto de árbol de derivación como de resultado del mismo.

Ejemplo 69 (Palíndromos, continuación) *La derivación*

$$P \Rightarrow_G 1P1 \Rightarrow_G 10P01 \Rightarrow_G 10101.$$

puede representarse mediante el siguiente árbol de derivación:



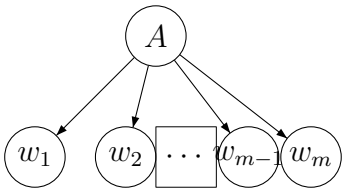
Nótese que la cadena obtenida concatenando, de izquierda a derecha, las hojas del árbol es, precisamente, 10101*.*

Obviamente, los árboles que centrarán nuestra atención son aquellos cuya raíz está etiquetada con la variable inicial y cuyas hojas están etiquetadas con carácteres terminales, i.e. representan cadenas del lenguaje.

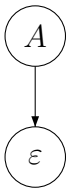
Veamos a continuación que los árboles de derivación y las secuencias de derivaciones son, esencialemnte los mismos objetos, i.e. son equivalentes.

Teorema 74 *Dada una gramática independiente del contexto, digamos* $G = (V, T, P, S)$*, una variable* $A \in V$ *y una cadena* $w \in T^*$ *tal que* $A \Rightarrow^* w$*. Entonces, existe un árbol de derivación cuya raíz está etiquetada con* A *y cuyo resultado es* w*.*

Demostración .– La demostración se lleva a cabo por inducción en el número de derivaciones.
Supongamos que w se puede inferir en un único paso de derivación a partir de A, i.e. que $A \to w$ es una producción. Si $w = w_1 \cdots w_m$, donde $w_1, \ldots, w_m \in (V \cup T)$, construimos el siguiente árbol de derivación



En el caso de que $w = \varepsilon$, el árbol que se construye es el siguiente:



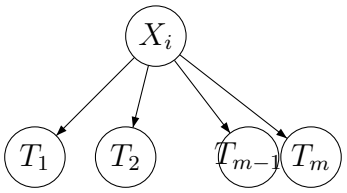
Por inducción, supongamos que para toda cadena derivable de A en a lo más n pasos, tiene asociada un árbol de derivación. Sea w una cadena que derivable a partir de A en $n+1$ pasos. Supongamos además que la primera derivación es de la forma $A \Rightarrow X_1 \cdots X_m$, donde $X_1, \ldots, X_m \in (V \cup T)^* \cup \{\varepsilon\}$. Para cada $i \in \mathbb{N}$, con $1 \leq i \leq m$, distinguimos los siguientes casos:

- Si $X_i = \varepsilon$, asociamos a X_i el árbol T_i definido como

- Si $X_i = w$, con $w \in T$, construimos el árbol de derivación

- Finalmente, si $X_i \in V$, considerando las derivaciones que afectan a X_i, obtenemos una cadena subcadena de w, digamos $w_i \in (V \cup T)^*$, y una secuencia de a lo más n derivaciones $X_i \Rightarrow w_i$. Por hipótesis inductiva, existe un árbol de derivación, digamos T_i, cuya raíz está etiquetada con X_i y cuyo resultado es, precisamente, w_i.

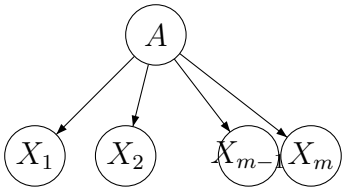
En resumen, para cada X_i, con $i \in \mathbb{N}$ y $1 \leq i \leq m$, tenemos un arból de derivación asociado T_i. Definimos entonces el árbol



La demostración concluye observando que el resultado de dicho árbol es la concatenación de los símbolos $w_1, \ldots, w_m$ (sean del alfabeto terminal o variables), i.e. $w = w_1 \cdots w_m$. ∎

Teorema 75 *Dado un árbol de derivación de una gramática $G = (V, T, P, S)$ cuya raíz está etiquetada mediante $A \in V$ y cuyo resultado es $w \in (V \cup T)^*$, existe una secuencia de derivaciones tales que $A \Rightarrow^* w$.*

Demostración .– La demostración se llevará a cabo por inducción en la profundidad[2] del árbol.
Supongamos que el árbol tiene profundidad nula. Por lo tanto, el árbol es de la forma:



donde $X_1, \ldots, X_m \in (V \cup T)^*$. Gracias a la definición de árbol de derivación se sigue que $A \to X_1 \cdots X_m$ es una producción, por lo que $A \Rightarrow X_1 \cdots X_m$ es una derivación admisible.
Supongamos que el resultado es cierto para cualquier árbol de derivación de profundidad n y demostremoslo para los árboles de profundidad $n+1$.
Sea A la raíz de un árbol de profundidad $n+1$, y sean $X_1, \ldots, X_k \in (V \cup T)$ las etiquetas de sus hijos. Si X_i es, para algún $i \in \mathbb{N}$ con $1 \leq i \leq k$, un carácter terminal, definimos $w_i := X_i$. En caso contrario, se sigue que X_i es una variable raíz de un árbol de profundidad a lo más n. Por hipótesis, existe una derivación que tiene como resultado el resultado del árbol cuya raíz es X_i, i.e. $X_i \Rightarrow^* w_i$. En cualquier caso, el resultado del árbol con raíz A es exactamente $w = w_1 \cdots w_k$, por lo que $A \Rightarrow X_1 \cdots X_k \Rightarrow^* w_1 \cdots w_k$. ∎

Nota 76 *Una atenta lectura de la demostración anterior revela que el tipo de derivaciones que aparecen son precisamente las derivaciones más a la izquierda. Esto es, dado un árbol de derivación podemos construir una derivación más a la izquierda con igual resultado.*

[2]Usualmente se define la profundidad de un árbol, o más generalmente de un grafo dirigido acíclico, como el máximo número de nodos intermedios que atraviesa un camino de la raíz a las hojas.

5.3. Autómatas a pila y gramáticas independientes del contexto

Una vez introducido el concepto de gramática independiente del contexto, a lo largo de la presente sección detallaremos su relación con la noción de autómata a pila.

Veamos, de entrada, que el lenguaje determinado por una gramática independiente del contexto cualquiera es aceptado por una autómata a pila que acepta por pila vacía.

Teorema 77 *Dada una gramática independiente del contexto $G = (V, T, P, S)$, existe un autómata a pila $N = (K, \Sigma, \Gamma, \delta, q_0, Z_0)$ que acepta por pila vacía tal que $L(G) = L(N)$.*

Demostración .– Supongamos dada la gramática G y construyamos un tal autómata N de la siguiente manera:

- K está formado por un único estado q (que por lo tanto es también estado inicial),
- el alfabeto de entrada coincide con el conjunto de símbolos terminales, mientras que el alfabeto de pila es la unión de los conjuntos de símbolos terminales y no terminales, i.e.

$$\Sigma = T \text{ y } \Gamma = V \cup T,$$

- el símbolo inicial de pila vendrá dado por el símbolo inicial de la gramática, y
- la función de transición se define como sigue:

$$\delta(q, \varepsilon, A) = \{(q, \beta) : A \to \beta \in P\},$$

cuando $A \in \Gamma$ se corresponde con una variable de la gramática G, y

$$\delta(q, a, a) = \{(q, \varepsilon)\}$$

en el caso de que a sea un carácter terminal de G.

Con el fin de demostrar que $L(N) \subseteq L(G)$, veamos en primer lugar que si en una secuencia de pasos el autómata extrae una variable de la pila consumiendo una cierta cadena, i.e.

$$(q, \omega, A) \vdash^* (q, \varepsilon, \varepsilon),$$

donde $A \in V$ y $\omega \in T^*$, podemos asociar a dicha secuencia una derivación en la gramática.
Para ello procedemos por inducción en la longitud de la secuencia de pasos de computación:

- (Caso Base) Supongamos, de entrada, que el autómata extrae una cierta variable A de la pila en un único paso. Por lo tanto, siendo $\delta(q, \omega, A) = (q, \varepsilon)$ se sigue que $\omega = \varepsilon$ y que $A \to \varepsilon \in P$.

- (Paso Inductivo) Si el autómata extrae una cierta variable A en n pasos de computación, puesto que A está en la cima de la pila, el primero de ellos debe ser el correspondiente a una transición asociada a una producción de la forma $A \to Y_1 \cdots Y_k$ (con $Y_i \in \Gamma$ para $1 \leq i \leq k$).

  Sea $w^i \in \Sigma^*$, con $1 \leq i \leq k$, la cadena consumida por el autómata para extraer de la pila el símbolo Y_i. Si $Y_i \in T$, para algún $1 \leq i \leq k$, forzosamente tenemos que $Y_i = w^i$. Si por el contrario $Y_i \in V$, para cierto $1 \leq i \leq k$, se sigue por hipótesis de inducción que $Y^i \Rightarrow^* w^i$, puesto que en ese caso tendríamos

$$(q, w^i, Y_i) \vdash^* (q, \varepsilon, \varepsilon).$$

En cualquier caso tenemos que $A \Rightarrow Y_1 \cdots Y_k \Rightarrow^* w^1 \cdots w^k$.
Recíprocamente, veamos que $L(G) \subseteq L(N)$. Probemos para ello que, dada una cadena $\omega \in T^*$ derivable a partir de una cierta variable X, existe una secuencia de pasos de computación que termina aceptando ω si en la pila se encontraba (exclusivamente) la variable X, i.e.

$$(q, \omega, X) \vdash^* (q, \varepsilon, \varepsilon).$$

Por inducción en el número de derivaciones a la izquierda necesarias para obtener ω a partir del símbolo inicial S.

- (Caso Base) Supongamos que una cadena $\omega \in \Sigma$ se deriva en una sola derivación a partir de cierta variable A, i.e. $A \Rightarrow \omega$. Puesto que ω pertenece a Σ^*, tenemos la siguiente secuencia de pasos de computación asociada:

$$(q, \omega, S) \vdash (q, \omega, \omega) \vdash^* (q, \varepsilon, \varepsilon),$$

  i.e. $\omega \in L(N)$.

- Supongamos que ω se deriva a partir de una variable A en n derivaciones más a la izquierda, digamos

$$A \Rightarrow \gamma_1 \Rightarrow \ldots \Rightarrow \gamma_n = \omega.$$

Puesto que hay más de una derivación, la cadena obtenida trás la primera derivación forzosamente contiene al menos una variable. Por construcción del autómata, los carácteres terminales que aparecen a la izquierda de las variables son procesados a la vez que los caracteres de la entrada mediante las transiciones del segundo tipo. En cuanto a las variable, cada vez que aparece una, puesto que se obtiene, a partir de ella, un fragmento de ω en a lo más $n-1$ derivaciones, se sigue por hipótesis inductiva que existe una secuencia de paso de computación que eliminan de la pila tanto la variable en cuestión como la subcadena de ω asociada.

Por lo tanto, dada $\omega \in L(G)$, existe una secuencia de pasos de computación que vacían la pila, i.e. $\omega \in L(N)$. ∎

TODO 9 *Completar la demostración anterior y dar un ejemplo.*

El recíproco también es cierto, tal y como muestra el siguiente Teorema.

Teorema 78 *Dado un autómata a pila $N = (K, \Sigma, \Gamma, \delta, q_0)$ que acepte por pila vacía, existe una gramática independiente del contexto $G = (V, T, P, S)$ tal que $L(G) = L(N)$.*

Demostración .– Construimos $G = (V, T, P, S)$ como sigue:

- el conjunto de variables se define

$$V = \{[pXq] : pq \in K, X \in \Gamma\} \cup \{S\},$$

donde S es el símbolo inicial,

- las producciones

$$S \to [q_0 Z_0 p]$$

para todo $p \in K$. Si $\delta(q, a, X)$ contiene $(r, Y_1 \cdots Y_r)$ para algún $a \in \Sigma \cup \{\varepsilon\}$ y $k \geq 0$, para toda lista $[r_1, \ldots, r_k]$ consideramos la producción

$$[qXr_k] \to [rY_1r_1] \cdots [r_{k-1}Y_kr_k]$$

pertenece a P.

∎

5.4. Ambiguedad en gramáticas y lenguajes

En la Sección anterior se ha visto que todo árbol de derivación tiene asociado una derivación (más a la izquierda). Sin embargo, en general no podemos concluir si dicha asociación es única, i.e. existen gramáticas para las que cadenas del lenguaje que generan pueden tener diversos árboles de derivación. Formalmente.

Definición 79 *Dada una gramática independiente del contexto, diremos que G es ambigua si existe una cadena en $L(G)$, el lenguaje generado por la gramática, tal que admite dos árboles de derivación. En caso contrario diremos que no es ambigua.*

Ejemplo 80 *El lenguaje de las expresiones aritméticas es ambiguo*

Un lenguaje generado por una gramática ambigua puede ser generado por otra gramática de manera tal que esta última no sea ambigua. Sin embargo, si toda gramática que genere un cierto lenguaje es ambiguo se dice que este es inherentemente ambiguo.

Además, la ambiguedad de las gramáticas es indecidible.

Los lenguajes empleados en la programación de computadores son no ambiguos.

Capítulo 6

Propiedades de los lenguajes independientes del contexto

6.1. Forma normal de Chomsky

El objetivo de está Sección es introducir el concepto de forma normal de Chomsky.

Definición 81 (Forma normal de Chomsky) *Una gramática $G = (V, T, P, S)$ se dice en forma normal de Chomsky si todas sus producciones son de la forma $A \to BC$ o $A \to a$ con $A, B, C \in V$ y $a \in T$.*

Teorema 82 *Todo lenguaje independiente del contexto (sin ε) admite una gramática en forma normal de Chomsky.*

Demostración .–

■

6.2. Lenguajes no independientes del contexto

6.3. Propiedades de clausura

6.4. Propiedades de decisión

6.5. Ejercicios

Capítulo 7

Máquinas de Turing

En este Capítulo se presenta el modelo de máquinas de Turing (véase [14]), discutiéndose, con cierta profundidad, algunos aspectos del mismo. Más concretamente, en la Sección 7.1 se define el concepto de máquina de Turing. También se introducen las nociones de lenguaje recursivo, recursivamente enumerables y enumerable. En la Sección 7.2 se presentan dos generalizaciones del concepto de máquinas de Turing: máquinas con múltiples cintas y máquinas no deterministas. En ambos casos se mostrará su equivalencia (computacional) con el modelo determinista de una única cinta. En la Sección 7.3 se da una breve introducción a la Teoría de la Complejidad (tanto en tiempo como en espacio). También se introduce el concepto de clases de complejidad y se discute la Conjetura de Cook. Finalmente, en la Sección 7.4 se discute la Tesis de Church–Turing.

La referencia fundamental a lo largo del presente Capítulo será [11].

7.1. Máquinas de Turing

Intuitivamente, una máquina de Turing es un dispositivo computacional que consta de una unidad de control (con un número finito de estados) y una unidad de lectura/escritura que se desplaza sobre una cinta, acotada por la izquierda, en la que se almacenan caracteres de un alfabeto finito. La unidad de lectura/escritura se desplaza a lo largo de la cinta alterando, si procede, el contenido de las celdas. Mientras que la unidad de control puede cambiar su estado en función de lo leído por la de lectura/escritura. La máquina termina su computación si llega a un estado de *parada*. Su definición formal es la siguiente.

Definición 83 (Máquina de Turing) *Una máquina de Turing es una cuadrupla* $M = (K, \Sigma, \delta, s)$ *tal que:*

1. *K es un conjunto finito de estados, suponemos que el llamado estado inicial s pertenece a K,*

2. *Σ es un alfabeto finito tal que $\Sigma \cap K = \emptyset$ y contiene los símbolos especiales $\sqcup$ y $\rhd$, llamados blanco y comienzo de cinta respectivamente,*

3. *$\delta : K \times \Sigma \longrightarrow (K \cup \{h, \text{“}si\text{”}, \text{“}no\text{”}\}) \times \Sigma \times \{\leftarrow, -, \rightarrow\}$ es una función, llamada de transición, tal que para todo $p, q \in K$ satisfaciendo $\delta(q, \rhd) = (p, \rho, D)$, se tiene $\rho = \rhd$ y $D = \rightarrow$.*

Obviamente, la noción anterior *estática* (no detalla como se llevan a cabo las computaciones). Para dotarla de *dinamismo* debemos describir cómo lleva a cabo la computación. Debemos definir para ello el concepto de configuración inicial y de paso de computación.

Definición 84 (Configuración) *Sea $M = (K, \Sigma, \delta, s)$ una máquina de Turing. Llamaremos configuración a toda tupla (q, v, w), donde $q \in K$ y $v, w \in \Sigma^*$. En particular, dada una cadena $x \in \Sigma^*$, a la tupla $(s, \rhd, x)$ se la llamará configuración inicial.*

Fijada la noción de configuración de una máquina de Turing, la transición entre las distintas configuraciones (paso de computación) se define como sigue.

Definición 85 (Paso de computación) *Sea $M = (K, \Sigma, \delta, s)$ una máquina de Turing y sean dos configuraciones $(q, v, w), (q', v', w') \in K \times \Sigma^* \times \Sigma^*$ tales que*

$$(q, v, w) = (q, v_1 \ldots, v_r, w_1 \ldots w_s), \text{ y}$$

$$(q', v', w') = (q', v'_1 \ldots v'_{r'}, w'_1 \ldots w'_{s'}).$$

Diremos que (q, v, w) da lugar a (q', v', w') en un paso, y lo denotaremos por $(q, v, w) \vdash_M (q', v', w')$, si suponiendo que $\delta(q, v_r) = (p, \sigma, D)$, se verifica que $q' = p$ y

- *si $D = \rightarrow$, $r' = r + 1$, $s' = s - 1$, $v'_i = v_i$ para $1 \leq i < r'$, $v'_{r'} = \sigma$, $w'_k = w_{k+1}$ para $1 \leq k \leq s'$,*

- *si $D = -$, $r' = r$, $s' = s$, $v'_i = v_i$ para $1 \leq i < r'$, $v'_{r'} = \sigma$, $w'_k = w_k$ para $1 \leq k \leq s'$, y*

- *si $D = \leftarrow$, $r' = r - 1$, $s' = s + 1$, $v'_i = v_i$ para $1 \leq i \leq r'$, $w_1 = \sigma$, $w'_k = w_{k-1}$ para $1 < k \leq s'$.*

Más generalmente, diremos que (q, v, w) *da lugar a* (q', v', w')*, y lo denotaremos por* $(q, v, w) \vdash_M^* (q', v', w')$*, si existe una secuencia finita de configuraciones* $(q_1, v_1, w_1), \ldots, (q_k, v_k, w_k)$*, para algún* $k \in \mathbb{N}$*, tal que*

$$(q, v, w) \vdash_M (q_1, v_1, w_1) \vdash_M \ldots \vdash_M (q_k, v_k, w_k) \vdash_M (q', v', w').$$

Una vez formalizada la parte *dinámica* de las máquinas de Turing, debemos interpretar los resultados que producen. Tres interpretaciones resultan de interés: *el reconocimiento de cadenas*, *el cálculo de funciones* y *la enumeración de conjuntos*.

Definición 86 (Aceptación/rechazo de cadenas) *Sea* $M = (K, \Sigma, \delta, s)$ *una máquina de Turing y* $x \in \Sigma^*$ *una cadena sobre el alfabeto* Σ*. Diremos que la máquina de Turing "para", si alcanza el estado "si", "no" o* h*, i.e. si* $(s, \triangleright, x) \vdash_M^* (q, v, w)$*, con* $q \in \{\text{"}si\text{"}, \text{"}no\text{"}, h\}$ *y* $v, w \in \Sigma^*$*. Si el estado final es "si", diremos que la máquina de Turing acepta* x*. Si el estado final es "no", diremos que la máquina de Turing rechaza la cadena* x*.*

Obviamente, tiene sentido hablar de los conjuntos reconocidos por la máquina de Turing. Bien es cierto que cabe hacer una distinción importante: una máquina de Turing puede aceptar un conjunto de cadenas y, bien rechazar las restante o bien permanecer computando indefinidamente. Esta distinción hace necesario introducir dos conceptos fundamentales: los lenguajes recursivos y los lenguajes recursivamente enumerables.

Definición 87 (Lenguajes recursivos) *Dados un lenguaje* $L \subseteq \Sigma^*$ *y una máquina de Turing* $M = (K, \Sigma, \delta, s)$*, diremos que la máquina de Turing* M *decide* L *si*

1. *para todo* $x \in L$*, entonces* $M(x) =$ "*si*"*, y*
2. *para todo* $x \notin L$*, entonces* $M(x) =$ "*no*"*.*

En este caso, diremos que L *es un lenguaje recursivo.*

En el caso anterior, la máquina de Turing *siempre* llega a una estado de parada, aceptando o rechazando las cadenas sobre un cierto alfabeto Σ. En general, esta situación no ocurre.

Definición 88 (Lenguajes recursivamente enumerables) *Dado un lenguaje* $L \subseteq \Sigma^*$ *un lenguaje y una máquina de Turing* $M = (K, \Sigma, \delta, s)$*, diremos que la máquina de Turing* M *acepta* L *si*

1. *para todo* $x \in L$, *entonces* $M(x) =$ "*si*", *y*
2. *para todo* $x \notin L$, *entonces* $M(x) =$ " $\nearrow$ " *(computa indefinidamente).*

En este caso, se dice que L *es un lenguaje recursivamente enumerable.*

El siguiente ejemplo muestra una máquina de Turing capaz de reconocer el lenguaje de los palíndromos. Aunque resulta un ejemplo sencillo, muestra la potencia de las máquinas de Turing frente a los autómatas finitos[1].

Ejemplo 89 (Palíndromos) *Consideramos* $M = (K, \Sigma, \delta, s)$ *la máquina de Turing, donde:*

- $\Sigma := \{0, 1, \triangleright, \sqcup\}$ *es el alfabeto,*
- $K := \{s, q_0, q_1, q_0', q_1', q\}$ *es el conjunto de estados, siendo* s *es el estado inicial, y*
- *la función de transición* δ *está definida mediante la tabla*

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$	$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	0	$(q_0, \triangleright, \rightarrow)$	q_0'	0	$(q, \sqcup, \leftarrow)$
s	1	$(q_1, \triangleright, \rightarrow)$	q_0'	1	("*no*", 1, $-$)
s	$\triangleright$	$(s, \triangleright, \rightarrow)$	q_0'	$\triangleright$	("*si*", $\triangleright$, $-$)
s	$\sqcup$	("*si*", $\sqcup$, $-$)	q_1'	0	("*no*", 0, $-$)
q_0	0	$(q_0, 0, \rightarrow)$	q_1'	1	$(q, \sqcup, \leftarrow)$
q_0	1	$(q_0, 1, \rightarrow)$	q_1'	$\triangleright$	("*si*", $\triangleright$, $-$)
q_0	$\sqcup$	$(q_0', \sqcup, \leftarrow)$	q	0	$(q, 0, \leftarrow)$
q_1	0	$(q_1, 0, \rightarrow)$	q	1	$(q, 1, \leftarrow)$
q_1	1	$(q_1, 1, \rightarrow)$	q	$\triangleright$	$(s, \triangleright, \rightarrow)$
q_1	$\sqcup$	$(q_1', \sqcup, \leftarrow)$			

Resulta sencillo observar que la máquina de Turing anterior acepta los palíndromos sobre el alfabeto $\{0, 1\}$ *y rechaza las restantes cadenas. En particular, el lenguaje de los palíndromos es recursivo.*

Además, es fácil ver que, para una entrada de longitud n*, el número de movimientos que realiza la máquina está acotadoen el caso peor por* $(n+1)n$.

En cuanto al cálculo de funciones mediante máquinas de Turing, su formalización se lleva a cabo en los términos siguientes:

[1] La relación entre ambos tipos de dispositivos abstractos de computación se dilucidará en los ejercicios del presente capítulo

Definición 90 (Cálculo de funciones) *Sea $M = (K, \Sigma, \delta, s)$ una máquina de Turing y $x, y \in \Sigma^*$ dos cadenas sobre el alfabeto Σ. Diremos que la máquina de Turing calcula y a partir de x si*

$$(s, \triangleright, x) \vdash_M^* (h, \triangleright, y).$$

Dada una función $f : D \subseteq \Sigma^* \to \Sigma^*$, diremos que la máquina de Turing M calcula f si se verifica:

1. $D := \{x \in M : M(X) =\downarrow \ \textit{(para)}\},\ y$
2. $f(x) = y$ *si y sólo si* $(s, \triangleright, x) \vdash_M^* (h, \triangleright, y)$.

Un ejemplo sencillo de esta situación es el cálculo del sucesor de un número expresado en binario.

Ejemplo 91 (Sucesor) *Consideramos $M = (K, \Sigma, \delta, s)$ definida como sigue:*

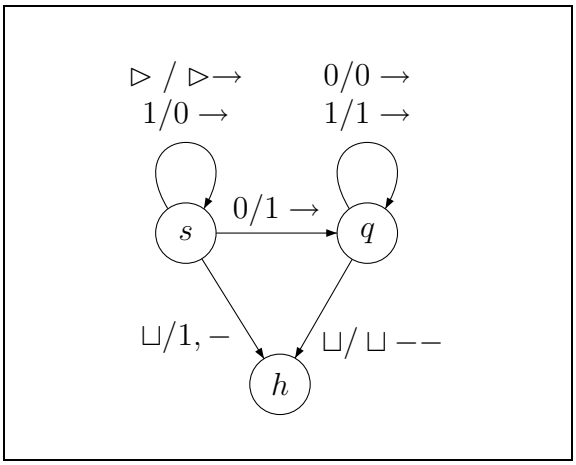
- $\Sigma := \{0, 1, \triangleright, \sqcup\}$ *es el alfabeto,*
- $K := \{s, q\}$ *es el conjunto de estados, siendo s es el estado inicial, y*
- *la función de transición δ está definida mediante la tabla*

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	0	$(q, 1, \rightarrow)$
s	1	$(s, 1, \rightarrow)$
s	$\sqcup$	$(h, 1, -)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
q	0	$(q, 0, \rightarrow)$
q	1	$(q, 1, \rightarrow)$
q	$\sqcup$	$(h, \sqcup, \leftarrow)$

Resulta sencillo observar que la máquina de Turing anterior calcula el sucesor de un número escrito en binario (suponemos los dígitos escritos en orden inverso).

El número de pasos que realiza la máquina, para una entrada de longitud n, está acotado por $n + 1$.

Por último, notar que la función de transición puede darse mediante diagramas de transición, al igual que en el caso de los autómatas. El correspondiente a la máquina anterior es:



Otro ejemplo sencillo es el cálculo del predecesor de un número.

Ejemplo 92 (Predecesor) *Consideramos $M = (K, \Sigma, \delta, s)$ definida como sigue:*

- $\Sigma := \{0, 1, \triangleright, \sqcup\}$ *es el alfabeto,*
- $K := \{s, q\}$ *es el conjunto de estados, siendo s es el estado inicial, y*
- *la función de transición δ está definida mediante la tabla*

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	0	$(s, 1, \rightarrow)$
s	1	$(q, 0, \rightarrow)$
s	$\sqcup$	$(\text{“}no\text{”}, 1, -)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
q	0	$(q, 0, \rightarrow)$
q	1	$(q, 1, \rightarrow)$
q	$\sqcup$	$(h, \sqcup, \leftarrow)$

Resulta sencillo observar que la máquina de Turing anterior calcula el predecesor de un número escrito en binario siempre que éste no sea nulo. Si el número en cuestión es cero, la máquina para en el estado “no”.

El número de pasos que realiza la máquina, para una entrada de longitud n, está acotado por $n + 1$.

Finalmente, podemos interpretar la computación llevada a cabo por una máquina de Turing como la enumeración de cierto lenguaje sobre un alfabeto finito Σ^*.

Definición 93 (Lenguaje Enumerable) *Sea $M = (K, \Sigma, \delta, s)$ una máquina de Turing y $L \subseteq \Sigma^*$ un lenguaje. Diremos que el lenguaje L es enumerable si la máquina de Turing M escribe en su cinta, de manera exhaustiva, todas y cada una de las cadenas de L (separándolas entre si mediante una marca).*

Postponemos un ejemplo de máquina de Turing del tipo anterior a fin de no complicar, más de lo necesario, el discurso.

Visto lo visto, queda claro que describir completamente una máquina de Turing, indicando el alfabeto, el conjunto de estados y la función de transición, es una tarea ardua e ingrata. Sin embargo, puede facilitarse teniendo en cuenta una serie de técnicas de programación. A continuación, desgranamos algunas de ellas a lo largo de diversos ejemplos.

El siguiente ejemplo ilustra como emplear los estados de una máquina, que aún constituyendo un conjunto finito, pueden emplearse para almacenar información sin más que escribir el estado como un par en el que una componente gestiona el control y la otra almacena la información.

Ejemplo 94 (Búsqueda de caracteres) *Sea $M = (K, \Sigma, \delta, s)$ la máquina de Turing definida como sigue:*

- $\Sigma := \{\sigma_1, \ldots, \sigma_n, \rhd, \sqcup\}$ *es el alfabeto,*
- $K := \{s\} \cup \{q_\sigma : \sigma \in \Sigma \setminus \{\rhd, \sqcup\}\}$ *es el conjunto de estados, siendo s el estado inicial, y*
- *la función de transición δ está definida mediante la tabla*

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	σ $(\sigma \in \Sigma \setminus \{\rhd, \sqcup\})$	$(q_\sigma, \sigma, \rightarrow)$
s	$\rhd$	$(s, \rhd, \rightarrow)$
s	$\sqcup$	$(\text{“}no\text{”}, \sqcup, -)$
q_σ $(\sigma \in \Sigma \setminus \{\rhd, \sqcup\})$	β $(\beta \in \Sigma \setminus \{\rhd, \sqcup\})$	$(q_\sigma, \beta, \rightarrow)$
q_σ $(\sigma \in \Sigma \setminus \{\rhd, \sqcup\})$	σ	$(\text{“}si\text{”}, \sigma, -)$
q_σ $(\sigma \in \Sigma \setminus \{\rhd, \sqcup\})$	$\sqcup$	$(\text{“}no\text{”}, \sqcup, -)$

Resulta sencillo ver que la anterior máquina de Turing acepta las cadenas cuyo primer carácter de la entrada aparece al menos otra vez. Para ello almacena en el estado el primer carácter sin más que saltar a un estado del tipo q_σ con $\sigma \in \Sigma \setminus \{\rhd, \sqcup\}$.

Otro problema frecuente es la gestión de la cinta. A continuación mostramos como se puede generar espacios en blanco o eliminar caracteres.

Ejemplo 95 (Inserción) *Sea $M = (K, \Sigma, \delta, s)$ la máquina de Turing definida como sigue:*

- $\Sigma := \{\sigma_1, \ldots, \sigma_n, \triangleright, \sqcup\}$ *es el alfabeto,*
- $K := \{s, \text{“}si\text{”}, \text{“}no\text{”}, h\} \cup \{q_\sigma : \sigma \in \Sigma \setminus \{\triangleright, \sqcup\}\}$ *es el conjunto de estados, siendo s es el estado inicial, y*
- *la función de transición δ está definida mediante la tabla*

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	σ $(\sigma \in \Sigma \setminus \{\triangleright, \sqcup\})$	$(q_\sigma, \sqcup, \rightarrow)$
s	$\sqcup$	$(h, \sqcup, -)$
q_σ $(\sigma \in \Sigma \setminus \{\triangleright, \sqcup\})$	β $(\beta \in \Sigma \setminus \{\triangleright, \sqcup\})$	$(q_\beta, \sigma, \rightarrow)$
q_σ $(\sigma \in \Sigma \setminus \{\triangleright, \sqcup\})$	$\sqcup$	$(h, \sigma, -)$

Resulta claro que M lleva a cabo la tarea anunciada.

Ejemplo 96 (Eliminación) *Sea $M = (K, \Sigma, \delta, s)$ la máquina de Turing definida como sigue:*

- $\Sigma := \{\sigma_1, \ldots, \sigma_n, \triangleright, \sqcup\}$ *es el alfabeto,*
- $K := \{s\} \cup \{q_\sigma : \sigma \in \Sigma \setminus \{\triangleright, \sqcup\}\}$ *es el conjunto de estados, siendo s es el estado inicial, y*
- *la función de transición δ está definida mediante la tabla*

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	σ $(\sigma \in \Sigma \setminus \{\sqcup\})$	$(s, \sigma, \rightarrow)$
s	$\sqcup$	$(q_\sqcup, \sqcup, \leftarrow)$
q_σ $(\sigma \in \Sigma \setminus \{\triangleright\})$	β $(\beta \in \Sigma \setminus \{\triangleright, \sqcup\})$	$(q_\beta, \sigma, \leftarrow)$
q_σ $(\sigma \in \Sigma \setminus \{\triangleright\})$	$\triangleright$	$(h, \triangleright, -)$

Resulta claro que M lleva a cabo la tarea anunciada.

7.2. Otras máquinas de Turings

En ocasiones, resulta útil considerar máquinas con varias cintas o que sean no deterministas. Comencemos definiendo las máquinas de Turing con multiples cintas.

Definición 97 (Máquina de Turing con k cintas) *Una máquina de Turing con k cintas es una tupla $M = (K, \Sigma, \delta, s)$ satisfaciendo:*

1. *K es un conjunto finito de estados, suponemos que el estado inicial s pertenece a K,*

2. *Σ es un conjunto finito de símbolos llamado alfabeto tal que $\Sigma \cap K = \emptyset$ y contiene los símbolos especiales $\sqcup$ y $\triangleright$, llamados blanco y comienzo de cinta respectivamente,*

3. *δ es una función, llamada de transición, definida del conjunto $K \times \Sigma^k$ en $(K \cup \{h, \text{“}si\text{”}, \text{“}no\text{”}\}) \times (\Sigma \times \{\leftarrow, -, \rightarrow\})^k$ tal que para todo $p, q \in K$, $\sigma_1, \ldots, \sigma_k, \sigma'_1, \ldots, \sigma'_k \in \Sigma$ y $D_1, \ldots, D_k \in \{\leftarrow, -, \rightarrow\}$ tales que $\delta(q, \sigma_1, \ldots, \sigma_k) = (p, (\sigma_1, D_1), \ldots, (\sigma_k, D_k))$, entonces para todo $i \in \mathbb{N}$, con $1 \leq i \leq k$, tal que $\sigma_i = \triangleright$ se verifica que $\sigma'_i = \triangleright$ y $D_i = \rightarrow$.*

Las definiciones de configuración y paso de computación son análogas a las del caso de máquinas de Turing deterministas.

En ocasiones, de entre las cintas de una máquina de Turing se distinguen dos: la cinta de entrada y la de salida. La cinta de entrada se considera sólo de lectura mientras que, la cinta de salida se considera sólo de escritura. Finalmente, notar que la definición de aceptación o rechazo en el caso de máquinas de Turing con múltiples cintas es idéntica al de las máquinas de Turing con una única cinta.

El siguiente ejemplo muestra como se simplifica el reconocimiento del lenguaje de los palíndromos mediante una máquina de Turing con dos cintas.

Ejemplo 98 (Palíndromos (y II)) *Consideramos $M = (K, \Sigma, \delta, s)$ la máquina de Turing, definida como sigue:*

- *$\Sigma := \{0, 1, \triangleright, \sqcup\}$ es el alfabeto,*

- *$K := \{s, q, q'\}$ es el conjunto de estados, s es el estado inicial, y*

- *la función de transición δ está definida mediante la tabla*

$p \in K$	$\sigma_1 \in \Sigma$	$\sigma_2 \in \Sigma$	$\delta(p, \sigma_1, \sigma_2)$
s	0	$\sqcup$	$(s, 0, \rightarrow, 0, \rightarrow)$
s	1	$\sqcup$	$(s, 1, \rightarrow, 1, \rightarrow)$
s	$\triangleright$	$\triangleright$	$(s, \triangleright, \rightarrow, \triangleright, \rightarrow)$
s	$\sqcup$	$\sqcup$	$(q, \sqcup, \leftarrow, \sqcup, -)$
q	0	$\sqcup$	$(q, 0, \leftarrow, \sqcup, -)$
q	1	$\sqcup$	$(q, 1, \leftarrow, \sqcup, -)$
q	$\triangleright$	$\sqcup$	$(q', \triangleright, \rightarrow, \sqcup, \leftarrow)$
q'	0	0	$(q', 0, \rightarrow, 0, \leftarrow)$
q'	1	1	$(q', 1, \rightarrow, 1, \leftarrow)$
q'	0	1	$(\text{“}no\text{”}, 0, -, 1, -)$
q'	1	0	$(\text{“}no\text{”}, 1, -, 0, -)$
q'	$\sqcup$	$\triangleright$	$(\text{“}si\text{”}, \sqcup, -, \triangleright, -)$

El número de movimientos que lleva a cabo la máquina en una entrada de tamaño n está acotado por $3n$.

Vista la noción de máquinas de Turing con varias cintas, podemos dar, de manera sencialla, un ejemplo de máquina que enumera un lenguaje.

Ejemplo 99 (Enumeración de los números naturales) *Sea M la máquina de Turing definida como sigue:*

- $\Sigma := \{0, 1, \triangleright, \sqcup\}$ *es el alfabeto,*
- $K := \{s, s', r, c, c'\}$ *es el conjunto de estados, s es el estado inicial, y*
- *la función de transición δ está definida mediante la tabla*

$p \in K$	$\sigma_1 \in \Sigma$	$\sigma_2 \in \Sigma$	$\delta(p, \sigma_1, \sigma_2)$
s	$\triangleright$	$\triangleright$	$(s, \triangleright, \rightarrow, \triangleright, \rightarrow)$
s	$\alpha \in \Sigma \setminus \{\triangleright\}$	$\beta \in \Sigma \setminus \{\triangleright\}$	$(s', 0, \rightarrow, 0, -)$
s'	$\alpha \in \Sigma$	$\beta \in \Sigma$	$(r, \#, \rightarrow, \beta, -)$
r	$\alpha \in \Sigma$	$\beta \setminus \{\triangleright\}$	$(r, \alpha, -, \beta, \leftarrow)$
r	$\alpha \in \Sigma$	$\triangleright$	$(c, \alpha, -, \triangleright, \rightarrow)$
c	$\alpha \in \Sigma$	0	$(c', 1, \rightarrow, 1, \rightarrow)$
c	$\alpha \in \Sigma$	1	$(c, 0, \rightarrow, 0, \rightarrow)$
c	$\alpha \in \Sigma$	$\sqcup$	$(s', 1, \rightarrow, 1, \rightarrow)$
c'	$\alpha \in \Sigma$	0	$(c', 0, \rightarrow, 0, \rightarrow)$
c'	$\alpha \in \Sigma$	1	$(c', 1, \rightarrow, 1, \rightarrow)$
c'	$\alpha \in \Sigma$	$\sqcup$	$(s', \alpha, \rightarrow, \sqcup, -)$

La función de los estados es la siguiente: s se encarga de inicializar un contador a cero (primer natural) y de escribirlo en la cinta de salida. s' se encarga de escribir la marca de separación entre naturales. c y c' se encargan de incrementar el contador copiando el nuevo natural en la cinta de salida. Finalmente, r se encarga de llevar el cursor al comienzo de la segunda cinta para incrementar posteriormente, empleando c y c', el contador.

Proposición 100 *Todo lenguaje aceptado (respec. decidido) por una máquina de Turing con k cintas es aceptado (respec. decidido) por una máquina de Turing.*

Posponemos la demostración de esta Proposición a la Sección 7.3, donde se considerarán, además, aspectos relativos a la Teoría de la Complejidad.

Veamos, por último, las máquinas de Turing no deterministas.

Definición 101 (Máquina de Turing no determinista) *Una máquina de Turing no determinista es una cuadrupla $M = (K, \Sigma, \Delta, s)$ tal que:*

1. *K es un conjunto finito de estados tal que $s \in K$,*
2. *Σ es un conjunto finito de símbolos llamado alfabeto tal que $\Sigma \cap K = \emptyset$ y contiene los símbolos especiales $\sqcup$ y $\triangleright$, llamados blanco y comienzo de cinta respectivamente,*
3. *$\Delta \subseteq (K \times \Sigma) \times ((K \cup \{h, \text{“}si\text{”}, \text{“}no\text{”}\}) \times \Sigma \times \{\leftarrow, -, \rightarrow\})$ es una relación tal que para todo para todo $q \in K$, si*

$$((q, \sigma), (p, \rho, D)) \in \Delta,$$

se tiene que $\sigma = \triangleright$, $\rho = \triangleright$ y $D = \rightarrow$.

Tanto la noción de configuración como la de computación de una máquina de Turing no determinista coinciden con las de una máquina determinista. En cuanto a la aceptación o el rechazo de entradas, éstas son aceptadas si alguna de las secuencias de pasos de computación llevan a un estado aceptador (“*si*”), y son rechazada en caso contrario, esto es, ninguna de las secuencias de computación llevan a un estado aceptador (“*no*”).

Ejemplo 102 (Palíndromos (y III)) *Sea $M = (K, \Sigma, \delta, s)$ la máquina de Turing definida como sigue:*

- *$\Sigma := \{0, 1, \triangleright, \sqcup\}$ es el alfabeto,*
- *$K := \{s, q, q'\}$ es el conjunto de estados, s es el estado inicial, y*

- *la función de transición Δ está definida mediante la tabla*

$p \in K$	$\sigma_1 \in \Sigma$	$\sigma_2 \in \Sigma$	$\Delta(p, \sigma_1, \sigma_2)$
s	0	$\sqcup$	$(s, 0, \rightarrow, 0, \rightarrow)$
s	1	$\sqcup$	$(s, 1, \rightarrow, 1, \rightarrow)$
s	0	$\sqcup$	$(s', 0, \rightarrow, \sqcup, \leftarrow)$
s	1	$\sqcup$	$(s', 1, \rightarrow, \sqcup, \leftarrow)$
s	0	$\sqcup$	$(q, 0, -, \sqcup, \leftarrow)$
s	1	$\sqcup$	$(q, 1, -, \sqcup, \leftarrow)$
s	$\rhd$	$\rhd$	$(s, \rhd, \rightarrow, \rhd, \rightarrow)$
s	$\sqcup$	$\sqcup$	$(\text{“}si\text{”}, \sqcup, -, \sqcup, -)$
q'	0	0	$(q', 0, \rightarrow, 0, \leftarrow)$
q'	1	1	$(q', 1, \rightarrow, 1, \leftarrow)$
q'	0	1	$(\text{“}no\text{”}, 0, -, 1, -)$
q'	1	0	$(\text{“}no\text{”}, 1, -, 0, -)$
q'	$\sqcup$	$\rhd$	$(\text{“}si\text{”}, \sqcup, -, \rhd, -)$

Obsérvese que, tanto el número de movimientos como el espacio consumido son menores que en el caso determinista.

Al igual que en casos anteriores, tenemos la equivalencia (en términos de aceptación o decisión de lenguajes) con las máquinas de Turing.

Proposición 103 *Todo lenguaje aceptado por una máquina de Turing no determinista es aceptado por una máquina de Turing (determinista).*

Postponemos la demostración de la Proposición anterior a la Seccion 7.3.

7.3. Introducción a la Complejidad

Resulta natural estudiar la eficacia de los algoritmos que se proponen para resolver un problema. Para ello, hay que definir de manera precisa el concepto de eficacia o, si se prefiere el de complejidad. Dos medidas de complejidad son utilizadas: el tiepo y el espacio. A lo lergo de esta Sección formalizaremos dichos conceptos, daremos algunos conceptos básicos y discutiremos la famosa conjetura de Cook.

7.3.1. Tiempo y espacio

Con respecto a la complejidad en tiempo, la definición es la que sigue:

Definición 104 (Complejidad en tiempo) *Sea $T : \mathbb{N} \longrightarrow \mathbb{N}$ una función y $M = (K, \Sigma, \delta, s)$ una máquina de Turing con k cintas. Diremos que la máquina de Turing opera en tiempo $T(n)$ si, para cada entrada de talla n, la máquina de Turing realiza, en el peor caso, $T(n)$ movimientos.*

Dada una función $T : \mathbb{N} \longrightarrow \mathbb{N}$ definida sobre los naturales, denotaremos por $TIME(T(n))$ el conjunto de los lenguajes decididos por máquinas de Turing deterministas en tiempo acotado por $T(n)$, esto es

$$TIME(T(n)) := \{L \subseteq \Sigma^* : L \text{ es decidido en tiempo } T(n)\}.$$

En lo que sigue, $TIME(T(n))$ recibe el nombre de clase de complejidad. Si las máquina de Turing son no deterministas, el conjunto de lenguajes decididos por máquinas de Turing no deterministas en tiempo acotado por $T(n)$, se denota mediante $NTIME(T(n))$.

De entre las clases de complejidad, dos resultan extremadamente populares: P y NP. Su definición es la siguiente:

$$P := \bigcup_{k \in \mathbb{N}} TIME(n^k), \text{ y}$$

$$NP := \bigcup_{k \in \mathbb{N}} NTIME(n^k).$$

Estas dos clases de complejidad han dado lugar a una de las conjeturas más famosas de la informática teórica: la conjetura de Cook (véase [4]). Dicha conjetura se enuncia como sigue:

Conjetura 105 (Cook) $P \neq NP$

Tras este enunciado sencillo, se esconde una cuestión tremendamente complicada. Tal es así, que el mecenas estadounidense Landon T. Clay, a través de su Clay Mathematics Institute, lo ha incluido en su celebrada lista de siete problemas. Su resolución está premiada con un millón de dolares.

En lo tocante al espacio, la definición es como sigue:

Definición 106 (Complejidad en espacio) *Sea $S : \mathbb{N} \longrightarrow \mathbb{N}$ una función y $M = (K, \Sigma, \delta, s)$ una máquina de Turing con k cintas ($k > 2$). Diremos que la máquina de Turing M opera en espacio acotado por $S(n)$ si para cada entrada de talla n la máquina de Turing emplea, en el peor caso, $S(n)$ celdas. Asumimos tácitamente que de entre las cintas de M, una de ellas está destinada a la entrada (sólo lectura) y otra a la salida (sólo escritura), por lo que no se contabilizan las celdas de la entrada en el cómputo de $S(n)$.*

Al igual que en el caso del tiempo, denotaremos por $SPACE(S(n))$ el conjunto de los lenguajes decididos por una máquina de Turing operando en espacio $S(n)$ (en el peor caso)

7.3.2. Resultados básicos

En esta Subsección demostramos cuatro resultados básicos: el Lema de aceleración lineal, el de compresión de cinta, el de reducción del número de cintas y el de eliminación de indeterminismo.

Lema 107 (Aceleración lineal) *Sea L un lenguaje aceptado por una máquina de Turing en tiempo acotado por $f(n)$, i.e. $L \in TIME(f(n))$. Entonces, para toda $\epsilon \in \mathbb{R}$ tal que $\epsilon > 0$, L es aceptado por una máquina de Turing en tiempo acotado por $\epsilon f(n) + n + 2$, i.e. $L \in TIME(\epsilon f(n) + n + 2)$.*

Demostración .– Sea $M = (K, \Sigma, \delta, s)$ una máquina de Turing, con una única cinta, que acepte el lenguaje L en tiempo acotado por $T(n)$. En lo que sigue, construiremos una máquina de Turing M' que acepta L en tiempo acotado por $\epsilon f(n) + n + 2$.
La idea de fondo es ampliar el alfabeto para codificar una mayor cantidad de información en un número menor de cintas y por tanto, necesitar menos movimientos para tratar la misma información. Para ello, consideramos el alfabeto $\Sigma' := \Sigma \cup \Sigma^k$, donde $k \in \mathbb{N}$ es un número a determinar que depende de ϵ.
La máquina M' tiene dos cintas: en la primera se encuentra la entrada codificada mediante el alfabeto Σ, mientras que en la segunda se simulará a M adecuadamente. De entrada, M' copia la cadena de entrada de su primera cinta a la segunda, codificándola en términos del alfabeto Σ', i.e. comprimiendola. Hecho esto, simula los movimientos de M hasta que ésta pare.
La simulación de los movimientos se lleva a cabo en bloques de k pasos de la siguiente manera: supongamos que la unidad de control de M' lee un cierto carácter $\sigma = (\sigma_1, \ldots, \sigma_k)$. La máquina M' lee los contenidos de las celdas situadas a izquierda y derecha de σ (3 pasos de M'). Puesto que en k pasos de M no abandonamos la región conformada por dichas celdas, podemos comprimir los k movimientos que M en a los más tres pasos de M' (actualizando el contenido de σ y de las celdas adyacentes). Si en algún momento M llega a un estado de parada, M' para de manera acorde.
La posición exacta, dentro de la secuencia σ, en la que se encuentra la unidad de lectura/escritura de M puede almacenarse en el estado al igual que el estado en el que se encuentra M.
Obviamente, el número de pasos de M', en el peor caso, es de

$$\left\lceil \frac{6f(n)}{k} \right\rceil .$$

Por lo tanto, tomando

$$k := \left\lceil \frac{6f(n)}{\epsilon} \right\rceil$$

se sigue el resultado. ■

La contrapartida, en lo referente al espacio, del anterior Lema es la siguiente:

Lema 108 (Compresión de cinta) *Sea L un lenguaje aceptado por una máquina de Turing en espacio acotado por $f(n)$, i.e. $L \in SPACE(f(n))$, y sea $\epsilon \in \mathbb{R}$, con $\epsilon > 0$. Entonces, L es aceptado por una máquina de Turing en espacio acotado por $\epsilon f(n)$, i.e. $L \in SPACE(\epsilon f(n))$.*

Demostración .– La demostración de este Lema se sigue de la del anterior. ■

Ambos Lemas consideran únicamente el caso de máquinas de Turing deterministas. Sin embargo, sigue siendo válidos en el caso de máquinas no deterministas.

Otro resultado interesante es el siguiente Lema que demuestra que todo lenguaje aceptado por una máquina con varias cintas es aceptado por una máquina con una única cinta.

Lema 109 (Reducción de cintas) *Sea L un lenguaje aceptado por una máquina de Turing con k cintas en tiempo acotado por $T(n)$. Entonces, L es un lenguaje aceptado por una máquina de Turing con una única cinta en tiempo acotado por $O(T(n)^2)$ (el espacio empleado no se ve alterado).*

Demostración .– Sea $M = (K, \Sigma, \delta, s)$ una máquina de Turing con k cintas que acepta L en tiempo acotado por $T(n)$. En lo que sigue, describiremos una máquina de Turing M' con una única cinta que acepte el lenguaje L.
La idea es codificar el contenido de las k cintas de M en la única cinta de M' sin más que escribir su contenido uno a continuación del otro. Para ello, definimos el alfabeto de M' como $\Sigma' := \Sigma \cup \{\underline{\sigma} : \sigma \in \Sigma\} \cup \{\triangleleft', \triangleright', \sqcup', \underline{\triangleleft}, \underline{\triangleright}, \underline{\sqcup'}\}$. Los símbolos subrayados se emplearán para indicar la posición de las unidades de lectura/escritura, mientras que $\triangleright'$ y $\triangleleft'$ se emplearán para indicar el comienzo y el fin de cada cinta.
Inicialmente, M' debe preparar el contenido de su cinta para llevar a cabo la simulación de M. Para ello, inserta símbolos los símbolo $\triangleright'$ y $\triangleleft'$ de manera adecuada. Para simular un movimiento de M, la máquina M' debe leer toda su cinta buscando caracteres subrayados, i.e. los caracteres asociados a las unidades de lectura/escritura de M). Dichos caracteres son almacenados en

los estados de M. La información del estado de M también se almacena en los estados de M'. Una vez recopilados los k caracteres subrayados, la máquina M' actualiza el contenido de su cinta de acuerdo a la función de transición de M, dicha función puede codificarse igualmente mediante los estado de M'. Queda claro que si alguna de las unidades de lectura/escritura de M sobreescriben algún carácter, lo único que tiene que hacer M' es actualizar el carácter subrayado conveniente, mientras que si se desplaza dicha unidad, lo único que debe hacer M' es subrayar el correspondiente carácter.
Puesto que la máquina M funciona en tiempo acotado por $T(n)$, a lo sumo escribe $T(n)$ caracteres en cada una de sus cintas, i.e. $kT(n)$ en total. Por lo tanto, M' tiene que dar del orden de $kT(N)$ pasos de computación para simular un único movimiento de M. De lo que se sigue que el número de pasos que da M' para simular M es del orden de $T(n)^2$. ∎

Finalmente, el siguiente resultado muestra que cualquier computación llevada a cabo por una máquina no determinista puede simularse en una determinista.

Lema 110 (Eliminación indeterminismo) *Supongamos que un cierto lenguaje L es aceptado por una máquina de Turing no determinista en tiempo acotado por $T(n)$ y sea c el grado de indeterminismo de M (máximo de las posibles elecciones que podemos hacer de acuerdo a Δ). Entonces, L es aceptado por una máquina de Turing determinista en tiempo acotado por $O(c^T(n))$.*

Demostración .– Sea $M = (K, \Sigma, \Delta, s)$ una máquina de Turing no determinista que decide L. Veamos como simular M mediante una máquina determinista M' con tres cintas. La funcionalidad de las cintas es la siguiente:

- En la primera cinta simplemente se almacena la entrada de la máquina M.
- En la segunda cinta, M' enumera secuencias finitas de enteros acotados por r (véase el Ejercicio **??**), dichos secuencias de números se corresponden con las posibles elecciones que tenemos de acuerdo a la relación δ. Nótese que no todas se corresponden con secuencias admisibles de computaciones puesto que no siempre tenemos r elecciones
- Por último, en la tercera cinta se lleva a cabo la simulación de M de acuerdo a las elecciones dadas por la última secuencia de números generada en la segunda cinta.

El funcionamiento de M' resulta ya claro. En primer lugar copia la entrada a la tercera cinta. Hecho esto, repite el siguiente proceso hasta finalizar la simulación: genera en la segunda cinta una secuencia de elecciones y simula en la tercera cinta dichos pasos hasta la finalización de la secuencia de pasos de computación asociados. La máquina M' acepta si alguna de las secuencias lleva a un estado aceptador en la simulación. ■

7.4. La Tesis de Church–Turing

Aunque uno disponga de una noción intuitiva de lo que es un algoritmo, su definición precisa exige rigor matemático. En concreto, podemos asimilar la noción de algoritmo con la de máquina de Turing. Sin embargo, dicha asimilación encierra una cierta arbitrariedad (quizás existen otros modelos de computación teóricos u otros modelos de computadoras aún por imaginar que sean capaces de llevar acabo una mayor cantidad de *tareas*). Afortunadamente, todos los modelos de computación estudiados hasta la fecha han resultado equivalentes al de las máquinas de Turing y todos los ordenadores concebidos por el hombre equivalentes a estas. Dicha coincidencia se conoce como la *Tesis de Church*, o *Church–Turing*. Dicha *tesis*, que postula que cualquier modelo de computación es equivalente a los anteriores, se encuentra sustentada en el hecho de que, hasta la fecha, así ha sido.

La formulación *matemática* de la *Tesis de Church–Turing* postula que la noción de *función computable* (algoritmo) puede identificarse con la de función recursiva (máquina de Turing). Obviamente, tal extremo no puede demostrarse aunque, tal y como hemos dicho, se sustenta en que así ha sido hasta la fecha.

A lo largo del curso veremos distintos modelos computacionales: las funciones recursivas introducidas por S.C. Kleene (véase [8]), el λ–calculus de A. Church ([3]) o los trabajos de E. Post (véase [12]). Para cada uno de ellos veremos su equivalencia con la noción de máquina de Turing sustentando de esta manera la tesis de Church.

Capítulo 8

Gramáticas sensibles al contexto y generales

A lo largo de esta Capítulo trataremos las gramáticas generales y las sensibles al contexto. Las primeras describen los lenguajes aceptables vía máquinas de Turing mientras que las segundas describen los lenguajes reconocidos por los autómatas linealmente acotados (máquinas de Turing no deterministas con espacio linealmente acotado por la longitud de la entrada), llamados sensibles al contexto.

Más precisamente, tras una breve introducción de las definiciones básicas (Sección 8.1), mostramos que todo lenguaje recursivamente enumerable está descrito por una gramática general y recíprocamente, que toda gramática general describe un lenguaje recursivamente enumerable. Por lo tanto, las propiedades de las gramáticas generales se siguen de las de los lenguajes recursivamente enumerable. Hecho esto, centramos nuestra atención en las gramáticas sensibles al contexto (Sección 8.2), demostrando los siguientes hechos: toda gramática sensible al contexto describe un lenguaje reconocible mediante un autómata linealmente acotado (y por ende, recursivo), todo autómata linealmente acotado acepta un lenguaje describible mediante una gramática al contexto y, finalmente, que existen lenguajes recursivos que no son sensible al contexto.

Finalizamos el Capítulo con el *Teorema de Jerarquía de N. Chomsky* que detalla la jerarquía de los lenguajes (o de las gramáticas si se prefiere).

La principal referencia a lo largo del presente Capítulo será [9], obra en la que hace hincapié tanto en las gramáticas como en los lenguajes, contrastando con la aproximación por nosotros seguida, hasta el momento, que incide en las máquinas.

8.1. Gramáticas no restringidas o generales

8.1.1. Definiciones básicas

Comencemos, sin más preámbulos, dando la definición de gramática general (comparar con las definiciones de las gramáticas regulares o independientes del contexto).

Definición 111 (Gramática general o no restringida) *Un gramática es una tupla $G = (N, T, P, S)$ donde*

- *N es un conjunto finito llamado alfabeto de caracteres no terminales,*
- *T es un conjunto finito, llamado de caracteres terminales, cuya intersección con el de no terminales es vacía, i.e. $N \cap T = \emptyset$,*
- *$P \subseteq (N \cup T)^+ \times (N \cup T)^*$ es una relación finita, a los elementos $(u, v) \in P$ se les conoce como producciones y se les denota mediante $u \to v$, y*
- *$S \in N$ es el símbolo de inicio.*

Siguiendo con el paralelismo existente con el caso de las gramáticas regulares o independientes del contexto, podemos definir la derivación en los siguientes términos.

Definición 112 (Derivación directa) *Sea $G = (N, T, P, S)$ una gramática general, $x, y \in (N \cup T)^*$ dos cadenas y $p = (u \to v) \in P$ una producción. Se dice que xvy deriva directamente de xuy de acuerdo a p y se denota mediante*

$$xuy \Rightarrow xvy \quad [p].$$

Nota 113 *Las notaciones $\Rightarrow^n$ (para $n \geq 0$), $\Rightarrow^*$, $\Rightarrow^+$ hacen alusión la aplicación de n derivaciones directas, un número indeterminado de ellas (quizás nulo) y un número indeterminado de ellas (al menos una) respectivamente.*

Teniedo en cuenta el concepto de derivación, podemos definir el lenguaje generado por una gramática general de la siguiente manera.

Definición 114 (Lenguaje generado por una gramática) *Sea $G = (N, T, P, S)$ una gramática general. Se define el lenguaje generado por G como el conjunto de todas las cadenas $w \in T^*$ que se derivan del carácter inicial S, i.e.*

$$L(G) = \{w \in T^* : S \Rightarrow^* w\}.$$

Obviamente, cualquier gramática vista hasta la fecha es, en particular, una gramática general.

8.1.2. Gramáticas generales y máquinas de Turing

A lo largo de la presente Sección mostraremos que los lenguajes aceptados por máquinas de Turing son, precisamente, los descritos por gramáticas generales. Para ello, demostramos en primer lugar que, dada una gramática general existe una máquina que acepta el lenguaje por ella descrito.

Teorema 115 *Sea $G = (N, T, P, S)$ una gramática general. Entonces, existe una máquina de Turing $M = (K, \Sigma, \delta, s)$ tal que $L(G) = L(M)$.*

Demostración .– Supongamos dada un gramática $G = (N, T, P, S)$ y construyamos una máquina de Turing $M = (K, \Sigma, \delta, s)$ que acepte el lenguaje $L(G)$.
Hacemos notar que $w \in T^*$ es una cadena del lenguaje $L(G)$ si y sólo si $S \Rightarrow^* w$ esto es, si podemos obtenerla por derivación del carácter inicial S. Por lo tanto, para construir una máquina que acepte el lenguaje $L(G)$ basta generar de manera ordenada las distintas cadenas del lenguaje $L(G)$ y compararlas con w. Los detalles se dan a continuación.
Definamos de entrada el alfabeto sobre el que trabaja la máquina de Turing M como la unión conjuntista tanto de los caracteres terminales y los no terminales como un conjunto de caracteres especiales propios a la operativa de la máquina de Turing $\{\triangleright, \sqcup, ;, \rightarrow\}$, i.e.

$$\Sigma = N \cup T \cup \{\triangleright, \sqcup, ;\}.$$

Además y para simplificar la exposición supondremos que la máquina de Turing tiene 4 cintas. La utilidad de las mismas es como sigue:

- En la primera cinta se almacena la entrada de la máquina de Turing, i.e. la cadena w.
- En la segunda cinta se almacenan las producciones de la gramática, i.e. se almacena una codificación de las mismas mediante los caracteres de Σ (emplearemos el carácter especial $\rightarrow$ para representar las producciones de forma natural y el carácter # para separar las descripciones de las mismas).
- La tercera cinta almacenará todas las cadenas que se derivan del carácter inicial S tras un cierto número de pasos. Dichas cadenas estarán separadas mediante el carácter de separación ;. Además, las cadenas obtenidas en n ($n \in \mathbb{N}$) derivaciones se separaran de las obtenidas en $n + 1$ derivaciones mediante la marca ##.

- La cuarta cinta está destinada a obtener las cadenas obtenidas en n derivaciones a partir de las cadenas obtenidas en $n-1$ derivaciones almacenadas en la tercera cinta y de las producciones almacenadas en la segunda.

En una primera instancia, la máquina escribe en la segunda cinta la codificación de las producciones de la gramática. De ello se encargan un conjunto finito de estados en los que se encuentra almacenada dicha información. Hecho esto, escribe en la tercera cinta el carácter de inicio S (interpretado como la cadena que se obtiene tras ninguna derivación) seguido de los correspondientes "##". Tras esta iniciación, la máquina genera de manera ordenada todas las derivaciones y comprueba si entre ellas se encuentra la cadena w. La generación de las derivaciones se lleva a cabo de manera iterativa. Con el fin de describir el proceso, supongamos que la máquina ha construido todas las cadena que se generan en n derivaciones a partir del carácter inicial S, que estas están almacenadas en la tercera cinta y que los últimos caracteres de dicha cinta son "##". La máquina itera como sigue:

- Sitúa la unidad de control de la tercera cinta tras el anterior "##", de esta forma la unidad de control queda al comienzo de las cadenas obtenidas en n derivaciones. Borra el contenido de la cuarta cinta sin más que sobrescribir caracteres $\sqcup$ (blancos) y sitúa la unidad de control de la segunda cinta (en la que se almacenan las producciones) al comienzo de la misma.

- Recorre la segunda cinta considerando una a una las distintas producciones. Para cada producción $u \to v \in P$, recorre las cadenas obtenidas en n iteraciones buscando la subcadena u. Cada vez que encuentra dicha subcadena, escribe en la cuarta cinta el resultado de aplicar dicha producción a la cadena en cuestión.

- Comprueba si alguna de las cadenas coincide con w. Si es el caso termina su ejecución aceptando w. En caso contrario, copia el contenido de la cuarta cinta al final de la tercera y vuelve a iterar.

Obviamente, la máquina de Turing para si y sólo si $w \in L(G)$, por lo que acepta dicho lenguaje. ■

Finalmente, para concluir la Sección, demostramos el siguiente Teorema que muestra que todo lenguaje aceptado por una máquina de Turing puede describirse mediane una gramática general; demostrando con ello lo anunciado al comienzo de la Sección.

Teorema 116 *Sea $M = (K, \Sigma, \delta, s)$ una máquina de Turing. Entonces, existe una gramática general $G = (N, T, P, S)$ tal que $L(M) = L(G)$.*

Demostración .– Supongamos dada una máquina de Turing $M = (K, \Sigma, \delta, s)$, construyamos $G = (N, T, P, S)$ una gramática general que tenga por lenguaje el aceptado por M.
Definimos, en primer lugar:

- el conjunto de los caracteres terminales como

$$T := \Sigma,$$

- el conjunto de los caracteres no terminales como

$$N := \{\langle\sigma\sigma'\rangle : \sigma, \sigma' \in \Sigma\} \cup K \cup \{S, A\}\},$$

  donde S y A son nuevos símbolos.

En cuanto al conjunto de las producciones P, lo definiremos de manera que cada subconjunto de producciones se encargue de simular una parte de la máquina de Turing.

- **(Inicialización)** Con el fin de simular la máquina de Turing M mediante la gramática general G, consideramos de entrada las siguientes producciones:

$$P := P \cup \{S \to S\langle\sqcup\sqcup\rangle, S \to A\langle\sqcup\sqcup\rangle, A \to s\},$$

  Para cada $\sigma \in \Sigma$ consideramos las siguientes producciones:

$$P := P \cup \{A \to A\langle\sigma\sigma\rangle\}.$$

  Tras aplicar una número finito de veces las producciones hasta aquí consideradas, obtenemos cadenas de la forma:

$$\langle\sigma_1\sigma_1\rangle \ldots \langle\sigma_m\sigma_m\rangle\langle\sqcup\sqcup\rangle^{(n)},$$

  donde $\sigma_1, \ldots, \sigma_m \in \Sigma$ y $\langle\sqcup\sqcup\rangle^{(n)}$ denota la cadena formada por n $\langle\sqcup\sqcup\rangle$. Por lo tanto, dichas producciones se encargan de generar la entrada de la máquina de Turing M así como los blancos necesarios para el funcionamiento de la misma.

- **(Simulación de movimiento de tipo $-$)** Supongamos que $\delta(\sigma, q) = (\sigma', q', -)$ con $q' \notin \{\text{“}si\text{”}, \text{“}no\text{”}, \text{“}h\text{”}\}$. Para simular un paso de la máquina de Turing mediante un conjunto de producciones de G, consideramos

$$P := P \cup \{q\langle\phi\sigma\rangle \to q'\langle\phi\sigma'\rangle\}$$

  donde $\phi \in \Sigma$ (puede ser un carácter blanco $\sqcup$).

- **(Simulación de movimiento de tipo $\to$)** Supongamos que $\delta(\sigma, q) = (\sigma', q', \to)$ con $q' \notin \{\text{“}si\text{”}, \text{“}no\text{”}, \text{“}h\text{”}\}$, entonces

$$P := P \cup \{q\langle\phi\sigma\rangle \to \langle\phi\sigma'\rangle q'\}$$

  donde $\phi \in \Sigma$ (puede ser un carácter blanco $\sqcup$).

- **(Simulación de movimiento de tipo $\leftarrow$)** Supongamos que $\delta(\sigma, q) = (\sigma', q', -)$ con $q' \notin \{\text{“}si\text{”}, \text{“}no\text{”}, \text{“}h\text{”}\}$, entonces

$$P := P \cup \{\langle\tau\rho\rangle q\langle\phi\sigma\rangle \to q'\langle\tau\rho\rangle\langle\phi\sigma'\rangle\}$$

  donde $\phi, \tau, \rho \in \Sigma$.

- **(Finalización)** Para cada estado de la forma $f \in \{\text{“}si\text{”}, \text{“}no\text{”}, h\}$ y cada par $\langle\tau\rho\rangle$ consideramos

$$P := P \cup \{\langle\tau\rho\rangle f \to f\tau f, f\langle\tau\rho\rangle \to f\tau f, f \to \lambda\}.$$

Obviamente, así definido, el conjunto de producciones de la gramática G es finito. Veamos que $L(G)$ coincide con $L(M)$. Para ello supongamos dado una entrada de la máquina M, digamos $\sigma_1, \ldots, \sigma_m \in \Sigma^*$, aceptado por la máquina de Turing. En este caso, la computación se lleva a cabo en un número finito de pasos y por ende, en un conjunto finito de celdas adicionales, digamos n. Puesto que la cadena

$$\langle\triangleright\triangleright\rangle\langle\sigma_1\sigma_1\rangle \ldots \langle\sigma_m\sigma_m\rangle\langle\sqcup\sqcup\rangle^{(n)}$$

se deriva directamente de S. Aplicando las producciones correspondientes a las computaciones llegamos, tras un número finito de pasos y más de $m + n$ caracteres, a un estado aceptador. Por lo tanto, aplicando las producciones de finalización queda que la cadena $\sigma_1 \ldots \sigma_m$ se deriva directamente de S.
En caso contrario, si la cadena $\sigma_1 \ldots \sigma_m$ no puede aceptarse en espacio a lo más $m + n$, obtendríamos una cadena con caracteres no terminales.
Resumiendo, la gramática anterior genera el mismo lenguaje que la máquina de Turing M. ∎

8.2. Gramáticas sensibles al contexto

A lo largo de la presente Sección consideraremos las gramáticas sensibles al contexto. Dichas gramáticas, tipo particular de las generales, describen lenguajes aceptado por autómatas linealmente acotados (máquinas de Turing no deterministas tales que el espacio que emplean es lineal en el tamaño de la entrada). Nuestro objetivo será mostrar que definen una clase intermedia de lenguajes, situada entre los independientes del contexto y los recursivos.

8.2.1. Gramáticas sensibles al contexto y autómatas linealmente acotados

Comenzamos esta Subsección definiendo el concepto de gramática sensible al contexto.

Definición 117 (Gramática sensible al contexto) *Una gramática general $G = (N, T, P, S)$ se dice sensible al contexto si para toda producción $u \to v \in P$ se verifica que $|u| \leq |v|$ (donde $|\cdot|$ denota la longitud de una cadena).*

Al igual que en ocasiones anteriores, podemos asociar a toda gramática sensible al contexto un lenguaje, llamado independiente del contexto.

Definición 118 (Lenguaje independiente del contexto) [1] *Un lenguaje $L \subset T^*$ se dice sensible al contexto si existe una gramática sensible al contexto G tal que $L = L(G)$.*

Las máquinas encargadas de reconocer tales lenguajes, tal y como se apuntaba en la introducción, son los autómatas linealmente acotados; definidos a continuación.

Definición 119 (Autómata linealmente acotado) *Un autómata linealmente acotado es una tupla $M = (K, \Sigma, \delta, s)$ tal que:*

1. *K es un conjunto finito de estados tal que $s \in K$,*
2. *Σ es un conjunto finito de símbolos llamado alfabeto tal que $\Sigma \cap K = \emptyset$ y contiene los símbolos especiales $\sqcap$, $\rhd$ y $\lhd$ (llamados blanco, comienzo y fin de cinta respectivamente).*

[1] Recordamos que un lenguaje $L \subseteq T^*$ se dice independiente del contexto si existe una gramática independiente del contexto G tal que $L = L(G)$

3. $\delta \subseteq (K \times \Sigma) \times ((K \cup \{h, \text{“}si\text{”}, \text{“}no\text{”}\}) \times \Sigma \times \{\leftarrow, -, \rightarrow\})$ *es una relación, llamada de transición, tal que*

   - *para todo* $p, q \in K$ *tales que* $(q, \triangleright, p, \rho, D) \in \delta$*, entonces* $\rho = \triangleright$ *y* $D = \rightarrow$*, y*
   - *para todo* $p, q \in K$ *y* $\rho \in \Sigma$ *tales que* $(q, \triangleleft, p, \rho, D) \in \delta$*, entonces* $\rho = \triangleleft$ *y* $D \in \{\rightarrow, -\}$*.*

Teniendo en cuenta las nociones de configuración de una máquina de Turing y paso de computación, podemos definir la dinámica de los autómatas linealmente acotados (basta observar que son un tipo particular de máquinas de Turing). Teniendo en cuenta éstas, podemos definir el lenguaje aceptado por una autómata linealmente acotado en los siguientes términos.

Definición 120 (Lenguaje aceptado por un autómata l. a.) *Un lenguaje* $L \subseteq \Sigma^*$ *es aceptado por un autómata* M*, si es reconocido por dicho autómata, i.e. si* M *acepta las cadenas de* L *y rechaza las restantes.*

Dichos lenguajes coinciden con los lenguajes sensibles al contexto previamente definidos. Para mostrar tal extremo, veamos en primer lugar que todo lenguaje generado por una gramática sensible al contexto es decidido por un autómata linealmente acotado.

Teorema 121 *Para toda gramática sensible al contexto* G *existe una autómata linealmente acotado* M *tal que* $L(G) = L(M)$*.*

Demostración .– Sea $G = (N, T, P, S)$ una gramática sensible al contexto, construiremos un autómata linealmente acotado con tres pistas tal que $L(G) = L(M)$.
Sea p el número de producciones de P y supongamos que toda cadena de T^* con longitud a lo más p es aceptada o rechazada de manera automática, i.e. que las cadenas aceptadas o rechazadas se codifican de alguna manera en los estados de M. Por lo tanto, podemos suponer que la entrada de M es una cadena de longitud mayor que p.
Inicialmente las pistas del autómata linealmente acotado contienen la siguiente información:

- la primera pista contiene la cadena de entrada $w \in T^*$ (suponemos que la longitud de w es mayor que p),
- la segunda cinta contiene una codificación de las producciones de P, para basta ampliar el alfabeto Σ, y por último

- la tercera cinta contiene el carácter S.

A partir de esta configuración inicial el autómata repite los siguientes cuatro pasos hasta que finaliza su ejecución:

- selecciona de manera no determinista una posición, digamos i, de la cadena de la tercera pista,
- selecciona de manera no determinista una producción a aplicar de entre las codificadas en la segunda pista, digamos $u \to v \in P$
- si $|w| \geq |x| + (|v| - |u|)$ y u es una subcadena de x con comienzo en el carácter i, entonces M deriva de x aplicando la producción $u \to v$. En caso contrario, rechaza w y para su ejecución.
- Si la tercera pista contiene w acepta, en caso contrario repite el presente proceso.

Resulta claro que el autómata anterior acepta el lenguaje generado por la gramática G. ∎

En cuanto al recíproco del anterior Teorema, basta observar que la demostración del Teorema 116 se aplica en el caso concreto de los autómatas linealmente acotados. Más precisamente, se tiene el siguiente resultado.

Teorema 122 *Para todo autómata linealmente acotado M existe una gramática sensible al contexto G tal que $L(M) = L(G)$.*

Demostración .– Supongamos dado un autómata linealmente acotado $M = (K, \Sigma, \delta, s)$. Para construir una gramática $G = (N, T, P, S)$ sensible al contexto que tenga por lenguaje el aceptado por M se procede como en la demostración del Teorema 116 con la salvedad de que tanto los estados como la marca de fin de cinta debe codificarse sobre el conjunto de símbolos no terminales puesto que la gramática debe ser sensible al contexto. ∎

Por lo tanto, se tiene la equivalencia entre autómatas linealmente acotados y gramáticas sensibles al contexto.

8.2.2. Gramáticas sensibles al contexto y lenguajes recursivos

Para concluir la Sección, resta demostrar que los lenguajes sensibles al contexto conforman una clase situada entre los recursivos y los independientes

del contexto. Resulta obvio que todo lenguaje independiente del contexto es sensible al contexto (basta observar las definiciones de las correspondientes gramáticas). Por lo tanto, a lo largo de esta Subsección nos centraremos en demostrar el contenido estricto de los sensibles al contexto en los recursivos.

Como primer paso, veamos que todo lenguaje sensible al contexto es recursivo.

Proposición 123 *Todo lenguaje sensible al contexto es recursivo.*

Demostración .– Para demostrar la afirmación tenemos que demostrar que dado un lenguaje sensible al contexto, existe una máquina de Turing que lo decide.
Sea pues L un lenguaje sensible al contexto, i.e. generado por una gramática sensible al contexto $G = (N, T, P, S)$. Construyamos una máquina de Turing no determinista con tres cintas que lo decida.
La utilidad de las cintas de la máquina de Turing es la siguiente:

- La primera cinta almacena la entrada $w \in T^*$. Puesto que a gramática es sensible al contexto, podemos suponer directamente que la longitud de la cadena de entrada es como poco 1 (en caso contrario la entrada sería la cadena vacía que no pertenece al lenguaje generado por una gramática sensible al contexto).

- La segunda contiene una codificación de las producciones de P. Al igual que en el Teorema 121, basta aumentar el alfabeto Σ sobre el que trabaja la máquina de Turing.

- La tercera cinta contiene, inicialmente, el carácter S.

Tomando como punto de partida las configuración anterior, la máquina opera de la siguiente manera:

- Supongamos que el contenido de la tercera pista viene dado por la cadena $\triangleright\ x$. La máquina selecciona, de manera no determinista, una posición, digamos i, de la cadena de la tercera pista.

- Selecciona de manera no determinista una producción almacenada en la segunda cinta, digamos $p : u \to v \in P$.

- Si la cadena x satisface $x = yuy'$ y la subcadena u comienza en la posición i, escribe en la tercera cinta, suponiendo que el contenido de la tercera cinta es de la forma $\triangleright\ wuw'\sqcup, \sqcap wvw'$.

- Si la tercera cinta contiene wvw' y $|wvw'| \geq |w|$ la máquina para rechazando la entrada.

■

Finalmente, veamos que el contenido es estricto, i.e. que existe un lenguaje recursivo que no es sensible al contexto.

Proposición 124 *Existe un lenguaje recursivo que no es sensible al contexto.*

Demostración .– Sea $\mathcal{N} = \{A_0, A_1, \ldots\}$ un subconjunto numerable infinito. En lo que sigue, supondremos que toda gramática $G = (N, T, P, S)$ satisface $N \subseteq \mathcal{N}$ y $S = A_0$ (esto no supone restricción alguna pues basta renombrar los elementos de N). Mediante un argumento similar al de la enumeración de las máquinas de Turing, podemos enumerar las gramáticas sensibles al contexto, digamos

$$G_1, G_2, G_3, \ldots$$

y las cadenas sobre el alfabeto $\{0, 1\}$, digamos

$$x_1, x_2, x_3, \ldots.$$

Notamos que a cada gramática G_i podemos asociarla una máquina de Turing que *decide* el lenguaje $L(G_i)$.
Consideramos el lenguaje

$$L_G := \{x_i : x_i \in L(G_i)\} \text{ y}$$

y su complementario L_G^c. En lo que resta de demostración veremos que L_G^c es recursivo pero no sensible al contexto.
Sea M la máquina de Turing M que tomando como entrada $w \in \Sigma^*$ procede de la siguiente manera

- Genera de manera ordenada los elementos de la sucesión $x_1, x_2, \ldots$ hasta encontrar el $i \in \mathbb{N}$ tal que $w = x_i$.

- A partir de dicho i, genera el código de la máquina de Turing asociada a la gramática sensible al contexto G_i, digamos M_i.

- Simula a la máquina M_i en la entrada w, aceptando si esta rechaza y viceversa.

Obviamente, la máquina de Turing anterior acepta el lenguaje L_G^c, por lo que este es recursivo. Gracias al argumento diagonal de Cantor podemos concluir que L_G^c no es sensible al contexto lo que concluye la demostración. ∎

De los enunciados anteriores se sigue el siguiente resultado.

Teorema 125 *La familia de los lenguajes recursivos contiene estrictamente a la familia de los lenguajes sensibles al contexto.*

Demostración .– Obvia a partir de los resultados anteriores. ∎

8.3. La jerarquía de Chomsky

Para concluir el Capítulo, damos el que se conoce como Teorema de la Jerarquía de Chomsky. Dicho Teorema establece las relaciones de contenido entre los diversos tipos de lenguajes que se han introducido a lo largo de los presentes apuntes. La demostración del mismo se sigue de los distintos contenidos que hemos ido viendo.

Teorema 126 (La jerarquía de Chomsky) *Se tienen las siguientes afirmaciones:*

1. *La familia de los lenguajes regulares está estrictamente contenida en la familia de los lenguajes independientes del contexto.*
2. *La familia de los lenguajes independientes del contexto (que no contienen la palabra vacía) esta estrictamente contenida en la familia de los lenguajes sensibles al contexto.*
3. *La familia de los lenguajes sensibles al contexto esta estrictamente contenida en la familia de los lenguajes recursivamente enumerables.*

Capítulo 9

Computabilidad

El objetivo fundamental del presente Capítulo es mostrar los limites de los actuales computadores. Teniendo en cuenta la equivalencia entre máquinas RAM (computadores) y máquinas de Turing, dicha cuestión se reduce a estudiar los limites computacionales de las máquinas de Turing, i.e. a estudiar las propiedades de los lenguajes recursivos y los recursivamente enumerables.

El esquema general que seguiremos es el siguiente:

En la Sección 9.1 estudiaremos algunas propiedades de los lenguajes, tanto recursivos como recursivamente enumerables, frente a operaciones conjuntistas (unión, intersección, complementario,...).[1].

En la Sección 9.2 se construye un lenguaje que no es recursivamente enumerable: el lenguaje diagonal. La no recursividad enumerables de dicho lenguaje implica la existencia de lenguajes que no son aceptados por máquina de Turing alguna. En términos coloquiales: existen problemas que no admiten solución algorítmica. Esencialmente el problema reside en la existencia de más lenguajes (problemas) que máquinas de Turing (algoritmos).

En la Sección 9.3 se construye un lenguaje recursivamente enumerable no recursivo: el lenguaje universal. Para ello, se detalla la construcción de una máquina de Turing capaz de simular a cualquier otra: la máquina de Turing universal. Dicha máquina, acepta un lenguaje recursivamente enumerable pero no recursivo: el lenguaje universal. El problema de fondo es que la máquina de Turing universal no puede determinar si la simulación que esta llevando a cabo concluirá o no. Como conclusión de la no recursividad del lenguaje universal se sigue que dentro de los lenguajes aceptados por máquinas de Turing podemos discernir dos categorías: los recursivos y los recursivamente enumerables (pero no recursivos). En términos coloquiales: dentro de los

[1] Dichas operaciones se verá complementadas, en los ejercicios, con algunas más propias de lenguajes como la concatenación.

problemas resolubles algorítmicamente, podemos distinguir aquellos que admiten exclusivamente una respuesta parcial (recursivamente enumerables) y aquellos que admiten una respuesta total (recursivos).

Continuando con el estudio de la recursividad y la recursividad enumerable, en la Sección 9 estudiaremos el Teorema de Rice. El objetivo de dicho teorema es mostrar que, las propiedades no triviales de los lenguajes no pueden determinarse algorítmicamente.

En la Sección 9.6 presentamos el problema de la correspondencia de Post como un nuevo ejemplo de lenguaje no recursivo. El interés de dicho problema, a parte del histórico, está en que permite demostrar fácilmente la intratabilidad de problemas relacionados con gramáticas, como por ejemplo: la ambiguedad gramáticas independientes del contexto.

El enfoque seguido es el de las obras [6] o [7]. Se invita a la lectura de dichos textos para información complementaria, ejemplos y ejercicios.

9.1. Propiedades básicas

Comencemos esta Sección estudiando el comportamiento de los *lenguajes recursivos* con respecto a operaciones conjuntistas básicas como la unión, intersección y complemento.

Teorema 127 *El complementario, la unión finita y la intersección finita de lenguajes recursivos es lenguajes recursiva.*

Demostración .– Veamos de entrada que el complementario de un lenguaje recursivo, digamos $L \subseteq \Sigma^*$, es un lenguaje recursivo.
Sea M una máquina determinista, con una cinta[2], que decida L. Construyamos una máquina N que decida L^c. Para ello, simplemente modificaremos la función de transición de M de manera que N acepte cuando ésta rechaza y viceversa. Está claro que N, así construida, acepta $w \in \Sigma^*$ sí y sólo si $w \notin L$, i.e. si $w \in L^c$.
La demostración relativa a la intersección se demuestra por inducción en el número de lenguajes a intersecar. Supongamos dados dos lenguajes recursivos $L_1, L_2 \subset \Sigma^*$ y sean M_1 y M_2 dos máquinas que los decidan. Construyamos una máquina M, con dos cintas, que decida $L_1 \cap L_2$. De entrada, M copia su entrada en la segunda cinta. Hecho esto, simula de manera simultánea a M_1 en la primera cinta y a M_2 en la segunda. Si las dos simulaciones terminan

[2]En vista del Lema de eliminación de indeterminismo y del Lema de reducción de cintas, podemos hacer tal suposición sin restricción de generalidad.

aceptando, M acepta; en caso contrario, M rechaza. Obviamente M'' decide $L_1 \cap L_2$.
Supongamos probada la afirmación para familias de k lenguajes recursivos, veamos que es cierta para familias de $k+1$ elementos. Sean $L_1, \ldots, L_{k+1}$ una familia de tales lenguajes. La afirmación se sigue de la asociatividad de la intersección de conjuntos, i.e. observando

$$L_1 \cap \ldots \cap L_{k+1} = L_1 \cap (L_2 \cap \ldots \cap L_{k+1}),$$

y de que $(L_2 \cap \ldots \cap L_{k+1})$ es un lenguaje recursivo al ser intersección de k lenguajes (hipótesis inductiva).
En cuanto a la unión, basta observar que, aplicando las leyes de Morgan, se tiene

$$L_1 \cup \ldots \cup L_{k+1} = (L_1^c \cap L_2^c \cap \ldots \cap L_{k+1}^c)^c.$$

■

En lo tocante a la unión e intersección finita de lenguajes recursivamente enumerables, la situación es completamente análoga a la del caso recursivo y se recoge en el siguiente Teorema.

Teorema 128 *La unión e intersección finita de lenguajes recursivamente enumerables es recursivamente enumerable.*

Demostración .– Veamos en primer lugar que la intersección de una familia finita de lenguajes recursivamente enumerables es recursivamente enumerables. Para ello, al igual que en el caso de los lenguajes recursivos, procederemos por inducción. Supongamos dados dos lenguajes recursivamente enumerable $L_1, L_2 \subseteq \Sigma^*$ y sean M_1 y M_2 las respectivas máquinas que los aceptan. Construimos M igual que en el caso de los lenguajes recursivos. Obviamante, $w \in \Sigma^*$ es aceptada por M si y sólo si ambas simulaciones paran aceptando. Por lo tanto, el lenguaje aceptado por M resulta ser $L_1 \cap L_2$. El caso general se sigue igual que en el caso de intersecciones de lenguajes recursivos.
En cuanto a la unión, procedemos igualmente por inducción. Supongamos dados dos lenguajes recursivamente enumerables $L_1, L_2 \subseteq \Sigma^*$ y sean M_1 y M_2 las respectivas máquinas que los acepten. Construimos una máquina M, con dos cintas, de la siguiente manera: de entrada, M copia su entrada de la primera cinta a la segunda; hecho esto, comienza a simula en paralelo a M_1 sobre la primera cinta y M_2 sobre la segunda. M acepta en el momento en que alguna de las dos simulaciones acepte[3]. Queda claro por tanto que M

[3]Nótese que en caso contrario, M permanece computando indefinidamente.

acepta $L_1 \cup L_2$. El caso general se sigue igual que los anteriores: observando la asociatividad de la unión y aplicando la hipótesis inductiva. ∎

Desafortunadamente el complementario de un lenguaje recursivamente enumerable no es necesariamente recursivamente enumerable. De hecho, se tiene el siguiente resultado.

Teorema 129 *Un lenguaje tal que él y su complementario son recursivamente enumerables es recursivo.*

Demostración .– Sea $L \subseteq \Sigma^*$ un lenguaje recursivamente enumerable tal que su complementario también es recursivamente enumerable. Sean M_1 y M_2 las máquinas que reconocen L y L^c respectivamente. Construimos una máquina M, con dos cintas, como sigue: de entrada, M copia su entrada de la primera a la segunda cinta; hecho esto, simula en paralelo la máquina M_1 en la primera cinta y M_2 en la segunda. Si la simulación de la máquina M_1 acepta, M acepta la entrada. Si por el contrario acepta la simulación de M_2, M rechaza la entrada[4]. Por lo tanto, M decide L, i.e. L es recursivo. ∎

Dado que, tal como veremos en la Sección 9.3, existen lenguajes recursivamente enumerables que no son recursivos, deducimos que existen lenguajes recursivamente enumerables cuyo complementario no es recursivamente enumerable.

9.2. Lenguajes no recursivamente enumerables

El objetivo de esta Sección es demostrar la existencia de lenguajes que no son recursivamente enumerables. La interpretación de este hecho, en términos más mundanos, es la existencia de más problemas (lenguajes) que algoritmos (máquinas de Turing). En consecuencia, existen lenguajes que no son reconocibles mediante máquinas de Turing, i.e. no admiten ni siquiera una respuesta parcial al problema del reconocimiento de los mismos. Equivalentemente, existen problemas para los que no hay algoritmos.

[4]Obsérvese que ambas máquinas (M_1 y M_2) no pueden aceptar la misma entrada pues aceptan lenguajes complementarios.

9.2.1. Codificación de máquinas de Turing

De acuerdo a los Ejercicios **??** y **??**, podemos asignar a cada cadena $w \in \Sigma^*$ un número natural[5]. El objetivo de esta Subsección es demostrar un resultado similar para el caso de máquinas de Turing, i.e. podemos asignar a cada natural una máquina de Turing (de manera única).

Para representar de tal manera una máquina de Turing debemos, en primera instancia, codificar sobre alfabeto la descripción de los elementos básicos de la misma: función de transición, estado inicial,... En lo que sigue, supondremos que las máquinas están definidas sobre un cierto alfabeto finito Σ.

Codificaremos las máquinas de Turing sobre el alfabeto Σ mediante cadenas en sobre el alfabeto

$$\Gamma := \{\langle, \rangle, q, |, \leftarrow, -, \rightarrow\} \cup \Sigma.$$

Tácitamente asumimos que $\langle, \rangle, \triangleright, |, \leftarrow, -, \rightarrow \notin \Sigma$ puesto que, si no es el caso, podemos emplear símbolos alternativos.

Supongamos dada una máquina de Turing $M = (K, \Sigma, \delta, q_1)$ tal que

$$K = \{q_1, \ldots, q_n\} \text{ y}$$

$$\Sigma = \{\sigma_1, \ldots, \sigma_m\}.$$

Codificaremos cada q_i, con $1 \leq i \leq n$, mediante la cadena q^i (tantas repeticiones del símbolo q como indique el subíndice). Obsérvese que el estado inicial se corresponde con la cadena q (basta renombrar los estados para que así sea).

En lo relativo a la función de transición, debemos codificar informaciones del tipo:

$$\delta(q_i, \sigma_j) = (q_k, \sigma_l, d),$$

donde $q_i, q_k \in K$, $\sigma_j, \sigma_l \in \Sigma$ y $d \in \{\leftarrow, -, \rightarrow\}$. Dicha información se codificará, simplemente, mediante la cadena:

$$\langle q^i | \sigma_j | q^k | \sigma_l | d \rangle \in \Gamma^*.$$

Por lo tanto, para codificar δ, basta concatenar las codificaciones asociadas a cada uno de los posibles pares (q_i, σ_j), donde $1 \leq i \leq n$ y $1 \leq j \leq m$, i.e δ se codifica mediante la cadena

$$\langle \delta_{11} | \ldots | \delta_{n1} | \ldots | \delta_{1m} | \ldots | \delta_{nm} \rangle,$$

[5]Aún es más, se sigue que existe una máquina de Turing que lleva a cabo dicha asignación

donde cada δ_{ij}, con $1 \leq i \leq n$ y $1 \leq j \leq m$, codifica la información relativa a $\delta(q_i, \sigma_j)$.

Obviamente, asuminedo esta codificación, se sigue el siguiente resultado:

Teorema 130 *Existe una máquina de Turing que decide si una cadena sobre* Γ *define una máquina de Turing o no, i.e. el lenguaje*

$$\{\langle M \rangle : M \text{ es una máquina de Turing}\}$$

es recursivamente enumerable (de hecho recursivo).

Por lo tanto, aplicando el Ejercicio **??**, se sigue que existe una máquina de Turing que numera el conjunto de las descripciones de máquinas de Turing, i.e. que tomando como entrada un número natural, produce como salida la descripción de una máquina de Turing (de manera biyectiva).

En resumen, fijado un alfabeto Σ, podemos hablar de la i–ésima cadena sobre el alfabeto o de la i–ésima máquina de Turing y por ende, identificar (o codificarlas si se prefiere) cada una de ellas con un natural.

9.2.2. El lenguaje diagonal

En esta Subsección presentamos el *lenguaje diagonal*, primer ejemplo de lenguaje no recursivamente enumerable que veremos. La interpretación de la no recursividad enumerable de L_d es clara: no existe un algoritmo capaz de reconocer L_d (siquiera de manera parcial), i.e. existen problemas que no admiten solución algorítmica.

La definición precisade tal lenguaje es la siguiente:

Definición 131 (El lenguaje diagonal) *Sea* Σ *un alfabeto finito, asumiendo la enumeración de cadenas y máquinas de Turing de la Subsección anterior, definimos el lenguaje diagonal como*

$$L_d := \{w_i \in \Sigma^* : w_i \notin L(M_i)\}.$$

La demostración formal de la no recursividad enumerable de L_d puede verse en el siguiente Teorema.

Teorema 132 *El lenguaje diagonal* L_d *no es recursivamente enumerable.*

Demostración .– Por reducción al absurdo, supongamos que el lenguaje L_d es recursivamente enumerable. Por lo tanto, existe una máquina de Turing M tal que acepta el lenguaje L_d. Puesto que M es una máquina sobre el alfabeto Σ, de acuerdo a la Subsección anterior, existe un número natural $i \in \mathbb{N}$ tal que M es la i–ésima máquina de Turing. Considerando la i–ésima cadena sobre el alfabeto Σ, $w_i \in \Sigma^*$, caben dos posibilidades:

- Si $w_i \in L_d$, se sigue, por la definición de L_d, que $w_i \notin L(M_i)$. Puesto que $L_d = L(M_i)$, llegamos a una contradicción.
- Si $w_i \notin L_d$, se tiene, de nuevo por la definición de L_d, que $w_i \in L(M_i)$. Al ser $L_d = L(M_i)$, llegamos de nuevo a un absurdo.

Por lo tanto, en ambos casos llegamos a una contradicción y por ende, L_d no puede ser recursivamente enumerable. ■

En resumen, el lenguaje diagonal no es un lenguaje no recursivamente enumerable, i.e. no admite respuesta algorítmica. En lo que sigue veremos otros ejemplos de problemas no recursivamente enumerables.

9.3. Lenguajes no recursivos

En la Sección anterior nuestro objetivo fue demostrar la existencia de lenguajes no recursivamente enumerables. En esta Sección, mostraremos que existen lenguajes recursivamente enumerables que no son recursivos, i.e. problemas qtales que cualquier algoritmo que para ellos se construya, sólo podrá dar respuesta en una parte de las entradas, en las otras permanecerá computando indefinidamente.

9.3.1. La máquina de Turing universal

En esta Sección construimos la conocida como máquina universal de Turing. *Grosso modo*, ésta toma como entrada la descripción de una máquina y de una entrada, simula el comportamiento de ésta sobre dicha entrada. La existencia de una tal máquina universal garantiza (varios años antes de la construcción de los primeros computadores) la existencia (al menos teórica) de dispositivos capaces de ejecutar cualquier algoritmo, toscamente hablando demuestra la existencia de los computadores.

Para construir la máquina de Turing, necesitaremos la codificación de máquinas de Turing introducida en la Sección anterior. Además, simularemos máquinas de Turing deterministas con una única cinta; esto no supone una restricción de generalidad puesto que toda máquina no determinista o con varias cintas es equivalente a una determinista con una única cinta (véanse los Lemas 109 y 110). Los detalles se recogen en el siguiente enunciado.

Teorema 133 (Existencia de la Máquina Universal) *Existe una máquina de Turing M_u, sobre el alfabeto*

$$\Omega := \Gamma \cup \{\triangleright, \underline{\triangleright}\} \cup \underline{\Sigma}$$

[6], *tal que tomando como entrada la descripción de una máquina de Turing* M *y una cadena* $w \in \Sigma^*$*, simula a* M *en la entrada* w*, aceptando si* M *acepta a* w*.*

Demostración .– Definimos M_u como una máquina de Turing sobre Ω con tres cintas. La utilidad de las cintas es la siguiente:

- En la primera cinta M_u almacena su entrada, digamos $\langle M, w\rangle$.
- En la segunda se almacena una representación de la cinta de M y de la posición de la unidad de lectura de M. Los caracteres subrayados de Ω se emplean para indicar la posición de la unidad de lectura/escritura de M, mientras que $\rhd$ se usa para simular el símbolo de inicio de cinta de M.
- Finalmente, en la tercera cinta se almacena el estado en el que se encuentra M de acuerdo a la siguiente codificación: se almacenan tantos caracteres q como correspondan al estado de M.

M_u funciona como sigue:

- De entrada, M_u escribe en la segunda cinta la cadena $\underline{\rhd} w$.
- Posteriormente, repite el siguiente proceso:
  - lee el carácter en la segunda cinta el carácter subrayado (que indica la posición de la unidad de lectura/escritura de M),
  - lee el estado en el que se encontraría M, i.e. el contenido de la tercera cinta,
  - con esta información, busca (en la primera cinta) el movimiento a realizar y lo lleva a cabo.

Si la simulación de M para, la máquina M_u para aceptando. ∎

Notamos que el lenguaje aceptado por la máquina de Turing universal puede definirse explícita de la siguiente manera:

Definición 134 (El lenguaje universal) *Sea* σ *un alfabeto finito, definimos el lenguaje universal (sobre* Σ*) como el conjunto de los pares descripción de máquina de Turing y entrada tales que dicha máquina para en una tal entrada, i.e.*

$$L_u := \{\langle M, w\rangle : M(w) = \text{“}si''\}.$$

[6] Γ es el alfabeto introducido en la Sección anterior.

En la siguiente Subsección omstraremos la no recursividad de este lenguaje.

En resumen, existe una máquina de Turing capaz de simular a cualquier otra máquina de Turing. En particular, dada una cadena sobre un alfabeto, la máquina de Turing Universal puede determinar si la máquina parará en una tal cadena.

9.3.2. No recursividad del lenguaje universal

El objetivo fundemantel de esta Subsección es, tal y como se anunció, mostrar la no recursividad del lenguaje universal, i.e. demostrar que no existe un algoritmo que decida dicho lenguaje.

Teorema 135 *El lenguaje universal L_u no es recursivo.*

Demostración .– Sea $w \in \Sigma^*$ una cadena cualquiera, podemos construir una máquina de Turing que tomando como entrada w devuelva como salida el natural $i \in \mathbb{N}$ que le corresponde. Empleando la máquina que enumera las máquinas de Turing, podemos producir, apartir de i, obtener una descripción de M_i. En resumen, existe una máquina de Turing N que tomando como entrada una cadena $w \in \Sigma^*$, produce la cadena sobre el alfabeto Γ (introducido en la Sección anterior) $\langle M, w\rangle$, donde M es la descripción de la i–ésima cadena (w en esta caso).
Si la cadena w pertenece al lenguaje diagonal, i.e. $w \in L_d$, se sigue que $N(w) = \langle M, w\rangle \in L_u^c$, puesto que la máquina M no para en la entrada w, i.e. $w \notin L(M)$.
Si por el contrario tenemos $w \notin L_d$, se sigue que $N(w) = \langle M, w\rangle \in L_u$, puesto que en ese caso M para en la entrada w, i.e. $w \in L(M)$.
Si L_u fuera recursivo, L_u^c sería recursivamente enumerable, i.e. existiría una máquina de Turing que aceptaría L_u^c. Componiendo la acción de N con una tal máquina de Turing, tendríamos que L_d es recursivamente enumerable, lo cual es absurdo. Por lo tanto, L_u^c no es recursivamente enumerable y por ende, L_u no es recursivo. ∎

Tal y comom se verá en lo que sigue, L_u no es el único lenguaje recursivamente enumerable no recursivo. Sin embargo, constituye un importante ejemplo y servirá para demostrar la no recursividad de otros lenguajes. Es por ello un prototipo de esta categoría de lenguajes.

9.4. Reducciones

En la demostración del Teorema 135, se demostró la no recursividad de L_u a partir de una reducción de L_d a L_u^c. La técnica allí empleada no es particular y está Sección está destinada a generalizarla. Comencemos definiendo lo que se entiende por reducción.

Definición 136 (Reducción) *Sean $L_1, L_2 \subseteq \Sigma^*$ dos lenguajes definidos sobre un cierto alfabeto Σ. Diremos que L_1 es reducible a L_2 (o se reduce a L_2) si existe una máquina de Turing que transforma cadenas de L_1 en cadenas de L_2 y cadenas de L_1^c en cadenas de L_2^c.*

Ejemplo 137 (Reducción de L_u a L_{ne}) *Sean $L_u, L_{ne} \subseteq \Sigma^*$ los lenguajes*

$$L_u := \{\langle M, w\rangle : M \text{ acepta } w\}$$

$$L_{ne} := \{\langle M\rangle : L(M) \neq \emptyset\}.$$

Veamos com reducir el lenguaje L_u al lenguaje L_{ne}. Para ello, dado una cadena $\langle M, w\rangle$, codificando una máquina de Turing M y una cierta cadena $w \in \Sigma^*$, definimos la máquina de Turing M' de la siguiente manera:

Dada una cadena $x \in \Sigma^*$, la máquina M' procede como sigue: de entrada simula a M en la entrada w, si M acepta w, M' acepta x.

Obviamente se tiene que

$$L(M') = \begin{cases} \emptyset & \text{si } M \text{ no acepta } w \\ \Sigma^* & \text{si } M \text{ acepta } w \end{cases}$$

Luego la transformación que a cada $\langle M, w\rangle$ asocia la máquina M' así construida es un reducción de L_u a L_{ne}.

Teniendo en cuenta lo anterior, estamos en condiciones de generalizar la técnica empleada en la demostración del Teorema 135.

Teorema 138 *Sean $L_1, L_2 \subseteq \Sigma^*$ dos lenguajes sobre Σ tales que exista una reducción de L_1 a L_2. Entonces,*

1. *si L_1 no es decidible (no es recursivo), L_2 no es decidible, y*

2. *si L_1 no es recursivamente enumerable, L_2 no es recursivamente enumerable.*

Demostración .– Sea M la máquina de Turing que da la reducción del lenguaje L_1 al lenguaje L_2.
Supongamos que L_1 no es decidible, i.e. recursivo. Si L_2 fuera decidible, existiría una máquina M_2 que lo decidiría, digamos M_2. Entonces, la composición de la reducción dada por M con la máquina de Turing M_2, decidiría el lenguaje L_1. Por lo tanto, L_2 no es decidible.
Análogamente, supongamos que L_1 no es recursivamente enumerable. Si L_2 fuera recursivamente enumerable, existiría una máquina M_2 que le aceptaría, digamos M_2. Entonces, la composición de la reducción dada por M con la máquina de Turing M_2 aceptaría el lenguaje L_1. Por lo tanto, L_2 no es recursivamente enumerable. ■

Siguiendo con el ejemplo anterior.

Ejemplo 139 (Reducción de L_u a L_{ne}, continuación) *Siguiendo con el Ejemplo 137, basta aplicar el Teorema precedente para concluir que L_{ne} no es recursivo (puesto que L_u no es recursivo).*

9.5. Teorema de Rice y propiedades de los lenguajes recursivamente enumerables

En lo que sigue, denotaremos por $\mathfrak{L}$ al conjunto de todos los lenguajes recursivamente enumerables, i.e.

$$\mathfrak{L} := \{L \subseteq \Sigma^* : \exists M \text{ t.q. } M \text{ acepta } L\}.$$

Interesandonos en los subconjuntos de $\mathfrak{L}$, introducimos el concepto de propiedad de los lenguajes recursivamente enumerable.

Definición 140 (Propiedad) *Llamaremos propiedad de los lenguajes recursivamente enumerables a todo subconjunto $\mathfrak{P} \subseteq \mathfrak{L}$. Además, diremos que una propiedad $\mathfrak{P}$ es trivial si es el subconjunto vacío o el total, i.e. si $\mathfrak{P} = \emptyset$ o $\mathfrak{P} = \mathfrak{L}$.*

Ejemplos de propiedades del lenguajes recursivamente enumerables son: ser recursivo, ser independiente del contexto,...

Puesto que a cada lenguaje recursivamente enumerable podemos asociarle la descripción de una máquina de Turing que lo acepte, podemos las propiedades como lenguajes. Más concretamente.

Definición 141 (Lenguaje asociado a una propiedad) *Dada un propiedad $\mathfrak{P}$, definimos el lenguaje asociado a la propiedad $\mathfrak{P}$, denotado por $L_{\mathfrak{P}}$, como el conjunto de las descripciones de máquinas de Turing que aceptan lenguajes de $\mathfrak{P}$, i.e.*

$$L_{\mathfrak{P}} := \{M : L(M) \in \mathfrak{P}\}$$

Desafortunadamente, dichos lenguajes (salvo para propiedades triviales) no son recursivos, tal y como demuestra el siguiente Teorema.

Teorema 142 (Teorema de Rice) *Toda propiedad no trivial de los lenguajes recursivamente enumerables es indecidible (equivalentemente, toda propiedad de los lenguajes recursivamente enumerables decidible es trivial).*

Demostración .– Supongamos que $\emptyset \notin \mathfrak{P}$ (si no es así, basta tomar su complementario $\mathfrak{P}^c$). Puesto que $\mathfrak{P}$ es no trivial, existe un lenguaje $L \in \mathfrak{P}$. Sea M_L la máquina de Turing que acepta L.
Supongamos que $L_{\mathfrak{P}}$ es decidible. Entonces existe una máquina $M_{\mathfrak{P}}$ que decide $L_{\mathfrak{P}}$. En lo que sigue construiremos una reducción del lenguaje universal L_u a $L_{\mathfrak{P}}$ empleando M_L.
Dada $\langle M, w\rangle$, la codificación de una máquina de Turing M y de una cadena $w \in \Sigma^*$, construimos la máquina M' como sigue: para una cierta entrada $x \in \Sigma$, la máquina M' simula a M para la entrada w. Si M acepta w, M' simula M_L en la entrada x, aceptando si esta lo hace.
Obviamente se tiene lo siguiente:

$$L(M') = \begin{cases} L & \text{si } M \text{ acepta } w \\ \emptyset & \text{si } M \text{ no acepta } w \end{cases}$$

Por lo tanto, si $\langle M, w\rangle \in L_u$ se sigue que $M' \in L_{\mathfrak{P}}$. En caso contrario, si $\langle M, w\rangle \notin L_u$, se verifica que $M' \notin L_{\mathfrak{P}}$ (basta observar que $\emptyset \notin \mathfrak{P}$).
Aplicando el Apartado 2 del Teorema 138 se sigue que $L_{\mathfrak{P}}$ no es recursivo, i.e. decidible. ∎

La situación, en el caso de las propiedades recursivamente enumerables, resulta más compleja y será estudiada a continuación. Más concretamente veremos que una propiedad $\mathfrak{P}$ es recursivamente enumerable (i.e. $L_{\mathfrak{P}}$ es recursivamente enumerable) si y sólo satisface las siguientes condiciones:

R1 Si $L \in \mathfrak{P}$ y $L' \subseteq L$, para algún lenguaje recursivamente enumerable L', entonces $L' \in \mathfrak{P}$.

R2 Si L es un lenguaje infinito de $\mathfrak{P}$, existe un subconjunto finito de L en $\mathfrak{P}$, y

R3 El conjunto de los lenguajes finitos de $\mathfrak{P}$ es enumerable, i.e. existe una máquina de Turing que enumera el conjunto de los lenguajes finitos de $\mathfrak{P}$.

Para ello, demostraremos previamente una serie de lemas técnicos.

Lema 143 *Si $\mathfrak{P}$ no satisface* **R1***, entonces $L_{\mathfrak{P}}$ no es recursivamente enumerable.*

Demostración .– Para demostrar el Lema reduciremos el lenguajes L_u^c al lenguaje $L_{\mathfrak{P}}$ (nótese que el lenguaje L_u^c no es recursivamente enumerable y el resultado se seguiría del Teorema 138).
Por hipótesis, existen lenguajes recursivamente enumerables $L_1 \subseteq L_2 \subseteq \Sigma^*$ tales que $L_2 \in \mathfrak{P}$ y $L_1 \notin \mathfrak{P}$ (emplearemos L_1 y L_2 en la construcción de la reducción).
La reducción toma como entrada un par $\langle M, w\rangle$, describiendo una máquina de Turing y una entrada, y produce como salida la descripción de una cierta máquina de Turing M'. La máquina M' funciona como sigue: en una entrada $x \in \Sigma^*$, realiza en paralelo las siguientes tareas:

1. Simula a la máquina M_1 en la entrada x.
2. Simula a la máquina M en la entrada w, y si dicha simulación termina, a M_2 en la entrada x.

La máquina M', así construida, satisface que

$$L(M') = \begin{cases} L_2 & \text{si } M \text{ acepta } w, \\ L_1 & \text{si } M \text{ no acepta } w, \end{cases}$$

Por lo tanto, $L(M')$ pertenece a $L_{\mathfrak{P}}$ si y sólo si M no acepta w. Aplicando el Teorema 138 se concluye el resultado. ∎

Lema 144 *Si existe un lenguaje infinito $L \in \mathfrak{P}$ tal que todo subconjunto infinito no pertenece a $\mathfrak{P}$, i.e. no satisface* **R2***, entonces $L_{\mathfrak{P}}$ no es recursivamente enumerable.*

Demostración .– Al igual que en la demostración del Lema anterior, reduciremos el lenguajes L_u^c al lenguaje $L_{\mathfrak{P}}$.
Por hipótesis, existe un lenguaje infinito $L_1 \in \mathfrak{P}$ tal que cualquier subconjunto infinito no pertenece a $\mathfrak{P}$ (emplearemos L_1 en la construcción de la reducción). En lo que sigue, sea $\hat{M}$ una máquina de Turing que acepte L.

La reducción toma como entrada un par $\langle M, w\rangle$, describiendo una máquina de Turing y una entrada, y produce como salida la descripción de una cierta máquina de Turing M'. La máquina M', dada una entrada $x \in \Sigma^*$, comienza simulando a la máquina $\hat{M}$ en x. Si el resultado de la simulación es positivo, i.e. $\hat{M}$ acepta x, simula a M en la entrada w durante $|x|$ movimientos. Si M no acepta a w tras $|x|$ movimientos, M' acepta la entrada x.
Si M acepta w tras exactamente j movimientos, tenemos que

$$L(M') = \{x \in L : |x| < j\},$$

por lo que $L(M') \notin \mathfrak{P}$. Por el contrario, si M no acepta w, se verifica que $L(M') = L \in \mathfrak{P}$. Aplicando el Teorema 138 se sigue el resultado. ∎

Lema 145 *Si $L_{\mathfrak{P}}$ es recursivamente enumerable, entonces el conjunto de los lenguajes finitos de $\mathfrak{P}$ es enumerable, i.e. satisface* **R3**.

Demostración .– Obviamente existe una máquina de Turing que enumera el conjunto el conjunto de los pares ordenados (i, j), con $i, j \in \mathbb{N}$, i.e. $\mathbb{N} \times \mathbb{N}$.
Suponiendo que $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$, podemos codificar cada σ_i mediante la cadena $10^{(i)}$. Por lo que podemos codificar los subconjuntos finitos de Σ^* mediante cadenas de ceros y unos, empleando la cadena 11 como separador entre palabras. Dado un número $i \in \mathbb{N}$ codificando un subconjunto finito, podemos escribir una máquina de Turing que acepta el lenguaje finito por él codificado.
Por hipótesis, $L_{\mathfrak{P}}$ es recursivamente enumerable. Por lo tanto, existe una máquina M que enumera las cadenas de $L_{\mathfrak{P}}$. Para demostrar el Lema, emplearemos M en la construcción de una máquina que enumere los lenguajes finitos de $\mathfrak{P}$.
La máquina que enumera los lenguajes finitos de $\mathfrak{P}$ repite los siguiente pasos:

1. Genera un par ordenado $(i, j) \in \mathbb{N} \times \mathbb{N}$,

2. Si el natural i codifica un lenguaje finito, construye la codificación de la máquina que reconoce el lenguaje determinado por i, digamos M^i.

3. Simula el comportamiento de M durante j pasos, si dicha simulación produce el código de la máquina M^i, este se escribe en la cinta correspondiente seguido de un carácter de separación.

4. En cualquier caso, se repite el proceso anterior.

Obviamente, la máquina anterior enumera los lenguajes finitos de $L_{\mathfrak{P}}$ con lo que concluye la demostración del resultado. ∎

Tras esta serie de Lemas estamos en disposición de demostrar el siguiente Teorema.

Teorema 146 *$L_{\mathfrak{P}}$ es recursivamente enumerable si y sólo si*

1. *Si $L \in \mathfrak{P}$ y $L' \subseteq L$, para algún lenguaje recursivo L', entonces $L' \in \mathfrak{P}$.*
2. *Si L es un lenguaje infinito de $\mathfrak{P}$, existe un subconjunto finito de L en $\mathfrak{P}$, y*
3. *El conjunto de los lenguajes finitos de $\mathfrak{P}$ es enumerable, i.e. existe una máquina de Turing que enumera el conjunto de los lenguajes finitos de $\mathfrak{P}$.*

Demostración .– Tras los Lemas anteriores resta probar que, si se verifican las condiciones anteriores, entonces $L_{\mathfrak{P}}$ es recursivamente enumerable. Para ello, construiremos explícitamente la máquina que lo acepta. Dicha máquina lleva a cabo las siguientes etapas:

1. Genera un par $(i, j) \in \mathbb{N} \times \mathbb{N}$.
2. Simula durante i pasos a la máquina enumeradora de los lenguajes finitos de $\mathfrak{P}$, la existencia de dicha máquina se ha visto en el Lema 145.
3. Si tras i, se ha generado un subconjunto finito completo, se simula la máquina M durante j pasos en cada una de las cadenas del subconjunto finito. Si no se ha generado un subconjunto finito completo se vuelve al primer paso.
4. Si la máquina M acepta en j pasos todas las cadenas del subconjunto finito, se acepta M. En caso contrario se vuelve al primer paso.

Veamos que el lenguaje aceptado por la anterior máquina de Turing es precisamente $L_{\mathfrak{P}}$. Supongamos que $L \in \mathfrak{P}$ y que M es una máquina de Turing que acepta L. Entonces, gracias a la condición 2, exista un lenguaje finito $L' \in \mathfrak{P}$ tal que $L' \subseteq L$. Si i es el número de pasos que le lleva a la máquina enumeradora de lenguajes finitos completar la escritura de L', y j es el número de pasos que le lleva a la máquina M aceptar las cadenas de L', concluimos que la máquina anterior aceptará M tras generar el par (i, j).
Recíprocamente, supongamos que la anterior acepta una cierta máquina M a partir de un par (i, j). Por lo tanto, existe un lenguaje finito L' de $L(M)$,

generado tras a lo más i pasos de la máquina enumeradora de lenguajes finitos, cuyas cadenas son aceptadas por M en, a lo más, j pasos. Por lo tanto, $L' \in \mathfrak{P}$ y $L' \subseteq L(M)$. De lo que se sigue, gracias a la Hipótesis 1, que $M \in L_{\mathfrak{P}}$. ∎

En resumen, ll Teorema de Rice implica que son indecidibles todos los problemas no triviales relacionados con los lenguajes aceptados por una máquina de Turing, como por ejemplo:

- Decidir si el lenguaje aceptado por una máquina es el vacío o no.
- Decidir si el lenguaje aceptado por una máquina es finito o no.
- Decidir si el lenguaje aceptado por una máquina es recursivo, independiente el contexto,...

Obviamente lo anterior no implica que cualquier cuestión sobre máquinas de Turing es indecidible. Por ejemplo, es decidible si una máquina efectuará un desplazamiento a la derecha o no.

9.6. El problema de la correspondencia de Post

En lo que sigue, estudiaremos uno de los problemas indecidibles más importantes: *el Problema de la correspondencia de Post* (véase [13], si bien nuestra aproximación esta tomada de [5]). Veremos como se aplica a problemas de decibilidad relativos a gramáticas.

9.6.1. Enunciado

Antes de nada, conviene fija claramente el enunciado del mismo.

Problema 1 (de la correspondencia de Post) *Sea Σ un alfabeto finito Σ y sean*

$$A = (a_1, \ldots, a_k) \text{ y } B = (b_1, \ldots, b_k).$$

dos listas de cadenas. La instancia (A, B) tiene solución si y sólo si existe una secuencia de naturales $i = (i_1, \ldots, i_m)$, $i_1, \ldots, i_m \in \mathbb{N}$ con $1 \leq i_j \leq k$ para $1 \leq j \leq m$, tales que

$$a_{i_1} \cdots a_{i_m} = b_{i_1} \cdots b_{i_m}.$$

Para ilustrar el problema, basta considerar el siguiente ejemplo.

Ejemplo 147 *Consideramos el alfabeto finito* $\Sigma = \{0, 1\}$ *y las listas* $A = (1, 100111, 10)$ *y* $B = (111, 10, 0)$. *La secuencia* $i = (2, 1, 1, 3)$ *es una solución del problema de la correspondencia de Post.*

Con el fin de demostrar la indecibilidad del problema de la correspondencia de Post, consideramos, en primera instancia la siguiente modificación del mismo.

Problema 2 (de la correspondencia de Post modificado) *Sea* Σ *un alfabeto finito y sean*

$$A = (a_1, \ldots, a_k) \text{ y } B = (b_1, \ldots, b_k).$$

dos listas de cadenas de Σ. *La instancia* (A, B) *tiene solución si y sólo si existe una secuencia de naturales* $i = (i_2, \ldots, i_m)$, $i_2, \ldots, i_m \in \mathbb{N}$ *con* $1 \leq i_j \leq k$ *para* $2 \leq j \leq m$, *tales que*

$$a_1 a_{i_2} \cdots a_{i_m} = b_1 b_{i_2} \cdots b_{i_m}.$$

Lema 148 *Si el problema de la correspondencia de Post fuera decidible, el problema de la correspondencia de Post modificado también lo sería.*

Demostración .– Consideremos una instancia del problema de la correspondencia de Post, i.e. un alfabeto finito Σ y dos sucesiones finitas de cadenas sobre Σ, digamos

$$A = (a_1, \ldots, a_k) \text{ y } B = (b_1, \ldots, b_k).$$

Definimos la siguiente instancia del problema de la correspondencia de Post modificado:
Suponiendo que los símbolos $\#, \$ \notin \Sigma$ dos nuevos símbolos y sea $\Sigma' := \Sigma \cup \{\#, \$\}$ un nuevo alfabeto. Sea a'_i la cadena obtenida a partir a_i insertando símbolos $\#$ tras cada carácter de a_i. Sea b'_i la cadena obtenida a partir de b'_i insertando el símbolo \$ antes de cada carácter de b_i. Definimos además las siguientes cadenas de caracteres:

$$a'_0 = \#a'_1 \quad b'_0 = b'_1$$

$$a'_{k+1} = \$ \quad b'_{k+1} = \#\$.$$

Finalmente, definimos las siguientes listas:

$$A' = (a'_0, a'_1, \ldots, a'_{k+1}) \text{ y}$$

$$B' = (b'_0, b'_1, \ldots, b'_{k+1}).$$

Obviamente, las listas A' y B' conforman una instancia del problema de la correspondencia de Pots (modificado). Además, la instancia dada por A' y B' tiene solución si y sólo si la instancia A y B admite una solución.
Para demostrar esta última afirmación, supongamos que $i_1, \ldots, i_r \in \mathbb{N}$ es una solución al problema de correspondencia de Post modificado con listas A y B. Puesto que a'_0 y b'_0 son las únicas cadenas que comienzan por el símbolo $\#$ y a'_{r+1} y b'_{r+1} son las únicas cadenas que finalizan por \$, tenemos que $i_1 = 0$ y $i_r = k + 1$. Finalmente, puesto que a'_0 coincide con a'_1 (salvo el primer carácter), y este esta construido a partir de a_1 y que b'_0 coincide con b'_1 (salvo el primer carácter), y este esta construido a partir de b_1, se sigue que $1, i_2, \ldots, i_r$ es una solución para el problema de la correspondencia de Post modificado. ■

9.6.2. Indecibilidad

En esta Subsección demostramos la indecibilidad del *Problema de la Correspondencia de Post*. Gracias al Lema 148, basta demostrar que el *Problema de la Correspondencia de Post Modificado* es indecidibles, tal y como afirma el siguiente Teorema.

Teorema 149 *El Problema de la Correspondencia de Post Modificado es indecidible, i.e. no recursivo.*

Demostración .– [7] Construyamos una reducción de L_u al problema de correspondencia de Post. Para ello, a partir de un par $\langle M, w\rangle$ debemos construir un instancia del problema de la correspondencia de Post, i.e. dos lista A, B sobre un cierto alfabeto.
Suponiendo que $M = (K, \Sigma, \delta, s)$, definimos $\Sigma' := \Sigma \cup \{\#\}$, donde $\#$ es un nuevo símbolo. La idea es construir un par de listas A, B de manera que si existe solución al problema de correspondencia esta sea de la forma $\#sw\#u_1q_1v_1 \ldots \#u_kq_kv_k$, donde

- (u_i, q_i, v_i), para $1 \leq i \leq k$, es una configuración de la máquina de Turing válida (sw representa la configuración inicial (q_0, u_0, v_0), donde $q_0 = s, u_0 = \lambda$ y $v_0 = w$),
- $(u_0, q_0, v_0), \ldots, k(u_i, q_i, v_k)$ es la secuencia de configuraciones de M en la entrada w, i.e.

$$(u_0, q_0, v_0) \vdash_M \ldots \vdash_M k(u_i, q_i, v_k),, \text{ y}$$

[7] Obsérvense las similitudes existentes entre esta demostración y la del Teorema 116.

- q_k es un estado aceptador, i.e. "*si*", "*no*" o h.

Para ello, construimos consideramos las listas A, B de cadenas sobre el alfabeto Σ' siguientes:

- *Grupo 0.-* Los primeros elementos de A y B son los siguientes:

Lista A	Lista B
$\#$	$\#sw$

  Simplemente se encargar de inicializar el proceso de simulación de la máquina de Turing mediante concatenación de cadenas.

- *Grupo 1.-* Las siguientes cadenas se encargan de copiar los fragmentos de las codificaciones de las configuraciones que no se ven afectados por la computación.

Lista A	Lista B	
α	α	para cada $\alpha \in \Sigma$
$\#$	$\#$	

- *Grupo 2.-* Las siguientes cadenas se encargan de simular el movimiento de la máquina de Turing, considerando para cada $p \in K$, $\alpha, \beta, \gamma \in \Sigma$, $q \in K \setminus \{\text{"}si\text{"}, \text{"}no\text{"}, h\}$ y las cadenas

Lista A	Lista B	
$q\alpha$	βp	si $\delta(q, \alpha) = (p, \beta, \rightarrow)$
$\gamma q\alpha$	$p\gamma\alpha$	si $\delta(p, \alpha, \leftarrow) = (p, \beta, \leftarrow)$
$q\#$	Γ	si $\delta(q, \sqcup) = (p, \alpha, \rightarrow)$
$\gamma q\#$	$p\gamma\beta\#$	si $\delta(q, \sqcup) = (p, \beta, \leftarrow)$

- *Grupo 3.-* Con el finalizar el proceso de simulación (si procede) consideramos, para cada $q \in \{\text{"}si\text{"}, \text{"}no\text{"}, h\}$ y $\alpha, \beta \in \Sigma$, las cadenas

Lista A	Lista B
$\alpha q\beta$	q
αq	q
$q\alpha$	q

y para cada $q \in \{\text{“}si\text{”}, \text{“}no\text{”}, h\}$

Lista A	Lista B
$q\#\#$	$\#$

En lo que sigue, llamaremos solución parcial del *problema de la Correspondencia de Post Modificado* con listas A y B, a todo par $(x, y) \in \Sigma^* \times \Sigma^*$ tal que

- x e y son concatenación de cadenas correspondientes de las listas A y B, y
- existe una cadena $z \in \Sigma^*$, que llamaremos resto, tal que $y = xz$.

Probemos por inducción que dada una secuencia válida de computaciones

$$(u_0, q_0, v_0) \vdash_M \ldots \vdash_M k(u_i, q_i, v_k),, \text{ y}$$

donde (u_i, q_i, v_i), para $0 \leq i \leq k$, es una configuración de la máquina de Turing (asumimos que (q_0, u_0, v_0), donde $q_0 = s, u_0 = \lambda$ y $v_0 = w$, representa la configuración inicial para la entrada w), existe una solución parcial de la forma

$$(x, y) = (\#u_0q_0v_0\# \ldots \#u_{k-1}q_{k-1}v_{k-1}\#, \#u_0q_0v_0\# \ldots \#u_kq_kv_k\#).$$

Notamos que esta es la única solución parcial cuya cadena más larga tiene como longitud $|y|$.
El caso $k = 0$ es trivial puesto que resulta obligado tomar los primeros elementos de las listas (*Grupo* 0), i.e. $(\#, \#sw)$ (equivalentemente tenemos la solución parcial $(\#, \#u_0q_0v_0)$).
Supongamos que la afirmación es cierta para un cierto $k \in \mathbb{N}$ y que $q_k \notin \{\text{“}si\text{”}, \text{“}no\text{”}, h\}$. El resto del par (x, y) es $z = u_kq_kv_k$. Los siguiente pares de cadenas a concatenar deben ser escogidos de manera que las cadenas de la lista A formen z, por lo que deben ser escogidos entre los *Grupos* 1 y 2. Por lo tanto, tras una serie de elecciones, obtenemos una nueva solución parcial de la forma $(y, yu_{k+1}q_{k+1}v_{k+1}\#)$ tal que

$$(q_k, u_k, v_k) \vdash_M (q_{k+1}, u_{k+1}, v_{k+1}).$$

En el caso $q_k \in \{\text{“}si\text{”}, \text{“}no\text{”}, h\}$, las únicas posibles elecciones de cadenas a concatenar son de los *Grupos* 1 *y* 3. Lo que demuestra la afirmación.
Por lo tanto, la máquina de Turing M para en la entrada w si y sólo si existe una solución al *Problema de la Correspondencia de Post Modificado*

con listas A y B. Lo que demuestra el resultado, puesto que si el *Problema de la Correspondencia de Post Modificado* fuera decidible, existiría un algoritmo capaz de decidir el lenguaje L_u. ∎

Gracias al Teorema y al Lema 148 se sigue la indecibilidad del *Problema de la correspondencia de Post.*

Corolario 150 *El programa de la correspondencia de Post es indecidible.*

9.6.3. Aplicaciones

Tal y como se apuntaba al comienzo de la Sección, el principal interés del *Problema de la Correspondencia de Post* reside en su aplicación a otros problemas, principalmente relacionados con gramáticas. A continuación presentamos la relativa a la indecibilidad de la ambigüedad de las gramáticas independientes al contexto.

Teorema 151 *Es indecidible cuando una gramática independiente del contexto arbitraria es ambigua.*

Demostración .– Sean

$$A = (a_1, \ldots, a_m) \text{ and } B = (b_1, \ldots, b_m)$$

dos listas de cadenas sobre un alfabeto finito Σ y sean $c_1, \ldots, c_m$ m símbolos nuevos. Consideramos los lenguajes

$$L_A := \{a_{i_1} \cdots a_{i_r} c_{i_r} \cdots c_{i_1} : i_1, \ldots, i_r \in \mathbb{N}\} \text{ y}$$

$$L_B := \{b_{i_1} \cdots b_{i_r} c_{i_r} \cdots c_{i_1} : i_1, \ldots, i_r \in \mathbb{N}\}.$$

Sea $G := (\{S, S_A, S_B\}, \Sigma \cup \{c_1, \ldots, c_m\}, P, S)$ la gramática independiente del contexto cuyas producciones vienen dadas por

$$\begin{aligned} P &:= \{S \to S_A, S \to S_B\} \\ &\cup \{S_A \to a_i S_A c_i : 1 \leq i \leq m\} \\ &\cup \{S_A \to a_i c_i : 1 \leq i \leq m\} \\ &\cup \{S_B \to b_i S_B c_i : 1 \leq i \leq m\} \\ &\cup \{S_B \to b_i c_i : 1 \leq i \leq m\}. \end{aligned}$$

Obviamente, $L(G) = L_A \cup L_B$.
Una instancia del problema de correspondencia de Post (A, B) tiene solución, digamos $i_1, \ldots, i_r$, si y sólo si se tiene

$$a_{i_1} \cdots a_{i_r} c_{i_r} \cdots c_{i_1} = b_{i_1} \cdots b_{i_r} c_{i_r} \cdots c_{i_1}.$$

Esto es, si y sólo si G es ambigua. ∎

Capítulo 10

Funciones recursivas parciales

El objetivo fundamental del presente Capítulo es determinar las funciones computables mediante máquinas de Turing.

Para ello, veremos de entrada algunos tipos sencillos de funciones que pueden computarse empleando máquinas de Turing (funciones iniciales) y con posterioridad, veremos como pueden combinarse para obtener funciones más complejas. De esta manera, determinamos ciertas clases de funciones computables mediante máquinas de Turing (funciones recursivas primitivas, funciones recursivas y funciones recursivas parciales). Hecho esto, mostraremos que cualquier función computable mediante una máquina de Turing cae en alguna de estas clases.

El lector interesado en el tema puede consultar [1, Capítulo 4], [10] o [2] para un tratamiento más detallado del asunto.

10.1. Funciones computables

A lo largo de esta Sección veremos algunas funciones computables básicas y como éstas pueden combinarse (efectivamente) para obtener nuevas funciones.

En lo que sigue, nos centraremos exclusivamente en funciones definidas sobre los naturales. Los motivos para ello son por un lado históricos: se remontan a los trabajos de S.C. Kleene (véase [8] por ejemplo), en los que definió la noción de algoritmo vía funciones recursivas parciales definidas sobre los naturales, y prácticos: al suponer las funciones definidas sobre los naturales logramos independencia del alfabeto de trabajo (cualquier cadena sobre un alfabeto finito puede codificarse mediante números naturales y por ende, cualquier función).

Con el fin de incidir en el hecho de que las máquinas de Turing computan

funciones, en ocasiones consideraremos máquinas de Turing con dos cintas especiales: la cinta de entrada y la cinta de salida (si ésta existe, i.e. si la función está definida para la entrada).

También asumiremos que las máquinas trabajan sobre el alfabeto binario, i.e. representaremos los números naturales en el alfabeto $\{0, 1\}$. Caracteres auxiliares como paréntesis u otros se asumirán directamente.

Una última observación: si bien las funciones que aparecerán apueden ser parciales, no prestaremos especial a su dominio de definición y consideraremos éstas haya donde estén definidas.

Comencemos, sin más dilación, el estudio de las funciones computables mediante máquinas de Turing. Más concretamente, veamos que funciones sencillas como la función constante e igual a cero, la función sucesor o las funciones proyección son computables.

Lema 152 (Computabilidad de funciones iniciales) *Las siguientes funciones son computables:*

- *La función constante e igual a cero, i.e.*

$$\begin{array}{rcl} 0 : \mathbb{N} & \longrightarrow & \mathbb{N} \\ n & \mapsto & 0. \end{array}$$

- *La función sucesor, i.e.*

$$\begin{array}{rcl} s : \mathbb{N} & \longrightarrow & \mathbb{N} \\ n & \mapsto & n+1. \end{array}$$

- *Las funciones proyección, i.e. dados $m, k \in \mathbb{N}$, con $n \geq k \geq 1$,*

$$\begin{array}{rcl} \pi_k^m : \mathbb{N}^m & \longrightarrow & \mathbb{N} \\ (n_1, \ldots, n_m) & \mapsto & n_k. \end{array}$$

Demostración .– Obviamente, la función constante cero es computada por una máquina de Turing que ignora su entrada y escribe 0 en la cinta de salida.
La función sucesor es computada por una máquina de Turing que incrementa en uno su entrada. Los detalles pueden consultarse en el Capítulo 7.
En lo tocante a las funciones proyección, basta considerar la máquina de Turing que recorre su entrada hasta dar con la i–ésima componente de la misma y copia ésta en la cinta de salida. Para ello, sólo debe llevar la cuenta las componentes leídas en una cinta auxiliar. ∎

En lo que sigue, las funciones citadas en el anterior Lema son las que emplearemos como piezas básicas en la construcción de funciones más complejas. Para ello, combinaremos efectivamente dichas funciones bien por composición, por recursión o por μ–recursión.

Veamos por ello, en primer lugar, que la composición de funciones computables es computable.

Lema 153 (Computabilidad de la composición de funciones) *Sean $g_1, \ldots, g_m, h$ funciones computables. Entonces, la composición, definida como*

$$f(\vec{x}) = h(g_1(\vec{x}), \ldots, g_m(\vec{x})),$$

es computable.

Demostración .– Supongamos dado un conjunto finito de funciones recursivas $g_1, \ldots, g_m, h$ computables mediante ciertas máquinas de Turing, digamos $M_{g_1} \ldots, M_{g_m}, M_h$. En lo que sigue, construiremos una máquina de Turing M_f que computa la función recursiva

$$f(\vec{x}) = h(g_1(\vec{x}), \ldots, g_m(\vec{x})).$$

Para ello, consideramos una máquina de Turing con $m + 3$ cintas, donde la utilidad de las mismas es la siguiente:

- la primera cinta es la de entrada y la última es la de salida,
- las cintas intermedias (de trabajo) son las cintas asociadas a $M_{g_1}, \ldots, M_{g_m}, M_h$.

El funcionamiento de M_f es como sigue: primeramente copia la entrada $\vec{x}$ a las m primeras cintas de trabajo y simula sobre las mismas a $M_{g_1}, \ldots, M_{g_m}$. Si las m simulaciones terminan, podemos suponer que en las cintas de trabajo aparecen exclusivamente las casdenas $g_1(\vec{x}), \ldots, g_m(\vec{x})$. Entonces, M_f escribe en la $m + 1$–ésima cinta de trabajo la cadena $(g_1(\vec{x}), \ldots, g_m(\vec{x}))$ y simula en la misma a la máquina M_h. Si dicha simulación termina, suponiendo que el contenido de la $m + 1$–ésima cinta es $h(g_1(\vec{x}), \ldots, g_m(\vec{x}))$, M_f copia dicho contenido en la cinta de salida y para. ∎

A parte de la composición, podemos recurrir, tal y como se dijo, a la recursión para construir nuevas funciones, como muestra el siguiente Lema.

Lema 154 (Computabilidad de la recursión primitiva de funciones) *Sean g, h funciones computables. Entonces, la recursión primitiva, definida mediante*

$$\begin{aligned} f(\vec{x}, 0) &= g(\vec{x}) \\ f(\vec{x}, y+1) &= h(\vec{x}, y, f(\vec{x}, y)), \end{aligned}$$

es computable.

Demostración .– Supongamos que las funciones f y g son computables mediante sendas máquinas de Turing, digamos M_g y M_h. La máquina de Turing que computa f, denotada M_f, es como sigue:

- M_f consta de cuatro cintas: la primera es de entrada, la segunda y la cuarta se encargarán de simular las cintas de M_g y M_h, y la cuarta es la cinta de salida de M_f.
- El alfabeto sobre el que trabaja M_f es el de M_g y M_h (o su unión si es necesario) junto con el símbolo especial $\underline{\triangleright}$. El uso de dicho símbolo es el siguiente: puesto que debemos calcular una función de modo recursivo, $\underline{\triangleright}$ indicará el comienzo de la cinta en las simulaciones de M_g y M_h, protegiendo de esta manera los datos de las computaciones en curso.
- En lo tocante al funcionamiento, M_f procede de la siguiente manera:
  - *Inicialización.-* Al comienzo, M_f tiene en su primera cinta la entrada, digamos $(\vec{x}, y)$. Si y es igual a 0, simplemente copia $\vec{x}$ en su segunda cinta y simula a M_g. Si la simulación de M_g termina, M_f copia el contenido de su segunda cinta a la cuarta (la cinta de salida de M_f) y para.
  - *Iteración.-* Si por el contrario se tiene que $y \neq 0$, M_f lleva a cabo el siguiente proceso: escribe al final de la segunda y tercera cinta el símbolo $\underline{\triangleright}$ para proteger el contenido actual de las cintas y escribe en la segunda cinta, tras dicho símbolo, la cadena $(\vec{x}, y-1)$. Hecho esto, pasa de nuevo a un estado inicial con el fin de calcular $f(\vec{x}, y-1)$ (se llama recursivamente) empleando la segunda cinta como cinta de entrada y la tercera como cinta de trabajo (a lo largo de dicho proceso, $\underline{\triangleright}$ desempeña el papel de marca de inicio de cinta). Si dicho proceso termina, en la tercera cinta de M_f, tras $\underline{\triangleright}$, podemos suponer que tenemos $f(\vec{x}, y-1)$. Finalmente, escribe en la segunda cinta la cadena $(\vec{x}, y, f(\vec{x}, y-1)$, borra la información restante (tras los símbolos $\underline{\triangleright}$) y simula M_h empleando la segunda cinta como cinta de entrada y la tercera como cinta de trabajo. Si dicha simulación concluye, podemos suponer que el contenido de la tercera cinta, tras el $\underline{\triangleright}$, es $h(\vec{x}, y, f(\vec{x}, y))$. M_f distingue entonces dos posibilidades:
    - Si a lo largo de las cintas sólo hay un $\underline{\triangleright}$, M_f copia $h(\vec{x}, y, f(\vec{x}, y))$ a la cuarta cinta y para su ejecución.
    - Si no es este el caso (M_f está en medio de una llamada recursiva), elimina los últimos símbolos $\underline{\triangleright}$ y devuelve el control a un estado fijado (punto de retorno).

■

Una última forma de construir nuevas funciones, a partir de otras dadas, es la μ–recursión. El siguiente Lema recoge los detalles a ella relativos.

Lema 155 (Computabilidad de la μ–Recursión de funciones) *Sea g una función computable. Entonces, la μ–recursion, definida mediante*

$$f(\vec{x}) = \mu y(g(\vec{x}, y) = 0) := \text{mín}\{y \in \mathbb{N} : g(\vec{x}, y) = 0\},$$

es computable.

Demostración .– Sea M_g la máquina que computa la función g. Construimos una máquina de Turing M_f que evalúa f como sigue:
Supondremos que M_f cuenta con cuatro cintas: la primera es la de entrada, la segunda simula la cinta de M_g, la tercera cinta es la cinta de trabajo de M_f y finalmente, la cuarta cinta es la de salida.
El funcionamiento de la máquina M_f es como sigue:

- *Inicialización.-* De entrada, M_f inicializa a cero un contador en su tercera cinta.
- *Iteración.-* M_f copia en la segunda cinta la cadena $(\vec{x}, k)$, donde k es el valor del contador de la tercera cinta y $\vec{x}$ es la entrada, y simula a M_g. Si el resultado de la simulación es 0, escribe k en la cinta de salida y para. Si el resultado de la simulación es distinto de cero, incrementa en uno el contenido de la tercera cinta y repite el proceso. Hacemos notar que las simulaciones de M_g pueden no terminar, quedando f no definida en la entrada $\vec{x}$.

■

Teniendo en cuenta lo anterior, funciones iniciales y operadores de construcción, introducimos diversas clases de funciones atendiendo a los operadores empleados. La primera de ella es la de las funciones recursivas primitivas.

Definición 156 (Funciones recursivas primitivas) *La clase de las funciones recursivas primitivas es la menor clase de funciones que contienen las funciones iniciales y es cerrada bajo composición recursión primitiva.*

Una buena parte de las funciones usuales son recursivas primitivas. Veamos algunos ejemplos.

Ejemplo 157 (Suma) *La función suma* $suma : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *se define en los siguientes términos:*

$$\begin{aligned} suma(n, 0) &= n \\ suma(n, m+1) &= \mathcal{S}(n+m) \\ &= \mathcal{S} \circ \mathcal{P}_3^3(n, m, suma(n, m)). \end{aligned}$$

Puesto que suma se define por composición y recursión primitiva, se sigue que es una función recursiva primitiva.

Ejemplo 158 (Multiplicación) *A partir de suma, podemos definir la función mult de la siguiente manera:*

$$\begin{aligned} mult(n, 0) &= K_0^1(n) \\ mult(n, m+1) &= suma(n, mult(n, m)) \\ &= \mathcal{S} \circ \mathcal{P}_3^3(n, m, mult(n, m)). \end{aligned}$$

Tanto las funciones iniciales (función constante cero, sucesor y proyección) como los operadores de construcción (composición y recursión primitiva) responden a funciones y procesos usuales y pueden considerarse como computables. Sin embargo, el siguiente Ejemplo da una función total computable que no es recursiva primitiva.

Ejemplo 159 (Ackermann) *La función* $A : \mathbb{N}^2 \to \mathbb{N}$ *definida por las ecuaciones*

$$A(0, y) = y + 1,$$

$$A(x+1, 0) = A(x, 1), \ y$$

$$A(x+1, y+1) = A(x, A(x+1, y)),$$

no recursiva primitiva.

Por lo tanto, resulta usual considerar las que se conocen como funciones recursivas (totales).

Definición 160 (Función recursiva) *La clase de las funciones recursivas es la menor clase de funciones que contiene a las funciones iniciales y es cerrada bajo composición, recursión primitiva y* μ*–recursión restringida (definida globalmente).*

En la Definición anterior falta precisar el concepto μ–recursión restringida: se dice que una función total se obtiene mediante μ–recursión restringida a partir de otra total dada si la primera está definida para cualquier valor.

Cabe insistir en que la funciones recursivas son totales (están definidas para todo natural o para toda tupla de naturales). Puesto que las máquinas de Turing pueden computar funciones parciales (definidas en un subconjunto), es frecuente introducir las funciones recursivas parciales.

Definición 161 (Funciones recursivas parciales) *La clase de las funciones recursivas primitivas es la menor clase de funciones que contiene a las funciones iniciales y es cerrada bajo composición, composición primitiva y μ–recursión (no restringida).*

Insistimos, la diferencia con las funciones recursivas está en que las parciales no son necesariamente extensibles a funciones totales.

Finalmente, teniendo en cuenta los lemas anteriores, se sigue el siguiente resultado.

Teorema 162 *Las siguentes clases de funciones son computables vía máquinas de Turing:*

1. *las funciones recursivas primitivas,*
2. *las funciones recursivas (totales), y*
3. *las funciones recursivas parciales.*

Un último apunte, las máquinas que computan funciones recursivas primitivas o recursivas (totales) siempre terminan su ejecución, no siendo este el caso en las funciones recursives parciales.

10.2. Máquinas de Turing y funciones

Si bien a lo largo de la Sección anterior mostramos que toda función recursiva parcial es computable vía máquinas de Turing, en esta Sección mostraremos el recíproco: toda máquina de Turing computa una función recursiva parcial. Veamos el mismo sin más dilación.

Teorema 163 *Todo función evaluada por una máquina de Turing es una función parcial recursiva.*

Demostración .– Sea $M = (K, \Sigma, \delta, q_1)$ una máquina de Turing que evalúa una cierta función $f : D \subseteq \mathbb{N} \to \mathbb{N}$, donde D es el dominio de definición de f, esto es

$$D := \{x \in \Sigma^* : M(x) = \downarrow\}.$$

Debemos demostrar la recursividad parcial de la función f.
De entrada codificamos los estados $K = \{q_1, \ldots, q_r\}$ mediante los enteros $1, \ldots, r$ y los símbolos de $\Sigma = \{\sigma_0, \ldots, \sigma_{s-1}\}$ mediante $0, 1, \ldots, s-1$ (suponemos que $\sigma_0 = \sqcup$). Teniendo en cuenta estas codificaciones, definimos las siguientes funciones:

$$mov(q, \sigma) := \begin{cases} 2 & \text{si } \delta(q, \sigma) = (q', \sigma', \rightarrow) \text{ para algún } q' \in K \text{ y } \sigma' \in \Sigma, \\ 1 & \text{si } \delta(q, \sigma) = (q', \sigma', \leftarrow) \text{ para algún } q' \in K \text{ y } \sigma' \in \Sigma, \\ 0 & \text{en cualquier otro caso.} \end{cases}$$

$$simb(q, \sigma) := \begin{cases} \sigma' & \text{si } \delta(q, \sigma) = (q', \sigma', d) \text{ con } q' \in K \text{ y } d \in \{\leftarrow, -, \rightarrow\}, \\ \sigma & \text{en caso contrario.} \end{cases}$$

$$estado(q, \sigma) := \begin{cases} \text{q} & \text{si } \delta(q, \sigma) = (q', \sigma', \rightarrow) \text{ para algún } q' \in K \text{ y } \sigma' \in \Sigma, \\ 1 & \text{si } \delta(q, \sigma) = (q', \sigma', \leftarrow) \text{ para algún } q' \in K \text{ y } \sigma' \in \Sigma, \\ 0 & \text{en cualquier otro caso.} \end{cases}$$

Las funciones anteriores toman un conjunto finito de valores. Además, los conjuntos en los que toma un cierto valor (constante) son conjuntos recursivos que vienen dados por igualdades. Por lo tanto, las funciones anteriores son funciones recursivas primitivas. Dichas funciones codifican la función de transición δ de la máquina de Turing M:

- $mov(q, \sigma)$ indica el desplazamiento que lleva a cabo la unidad de control de M en función del estado q en el que se encuentra y el carácter σ que está leyendo.
- $simb(q, \sigma)$ da el símbolo que escribe la máquina de Turing M cuando se encuentra en el estado q y está leyendo el carácter σ.
- $estado(q, \sigma)$ indica el estado al que pasa la máquina M cuando se encuentra en el estado q y está leyendo el carácter σ.

Con el fin de codificar en términos similares un paso de computación, debemos codificar las configuraciones instantáneas de la máquina de Turing $(q, u, v) \in K \times \Sigma^* \times \Sigma^*$ mediante naturales. Para ello, a cada $(q, u, v) \in K \times \Sigma^* \times \Sigma^*$ le asociamos la tupla $(q, w, n) \in \mathbb{N}^3$ construida del modo siguiente:

- $q \in \mathbb{N}$ es el natural que codifica el estado $q \in K$,

- $w \in \mathbb{N}$ es la codificación de la cadena uv, esto es si $uv = \sigma_0, \ldots, \sigma_t$, donde $\sigma_i \in \Sigma$ para $0 \leq i \leq t$, definimos

$$w := \sum_{i=0}^{t} \sigma_i s^{t-i},$$

  donde s es el número de elementos de Σ y $\sigma_i \in \{0, \ldots, s-1\}$, para $0 \leq i \leq t$, denota la codificación del símbolo σ_i.

- $n := |u|$ es la posición de la unidad de control en la cinta.

Codificando de esta manera la configuración instantánea de la máquina de Turing M, podemos definir las siguientes funciones

$$simbact(w, q, n) := coc(w, b^{n-1}) \div mult(b, coc(w, b^n)),$$

$$\begin{aligned} cabezasig(w, q, n) &:= n \div eq((mov(p, simbact(w, q, n), 1)) \\ &+ eq(mov(p, simbact(w, q, n)), 2)), \end{aligned}$$

$$estadosig(w, q, n) = estado(p, simbact(w, q, n) + mult(k, \not{c}abezasig(w, q, n)),$$

$$\begin{aligned} cintasig(w, q, n) &:= (w \div mult(b^n, siymact(w, q, n) \\ &+ mult(b^n, simb(p, simbact(w, q, n))), \text{ y} \end{aligned}$$

$$paso(w, q, n) = (cintasig(w, q, n), estadosig(w, q, n), cabezasig(w, q, n)).$$

Las funciones llevan a cabo las siguientes tareas:

- $simbact(w, q, n)$ toma la codificación mediante naturales de una configuración y devuelve el carácter que esta leyendo la unidad de control.

- $cabezasig(w, q, n)$ toma la codificación de la configuración actual y devuelve la codificación del movimiento a realizar por la máquina de Turing, i.e. 2 si se debe desplazar a la derecha, 1 si es a la izquierda.

- $estadosig(w, q, n)$ devuelve la codificación del estado al que pasa la máquina de Turing una vez llevado a cabo el paso de computación.

- $cintasig(w, q, n)$ devuelve un entero que codifica la cinta de la máquina de Turing una vez escrito el carácter pertinente.

- $paso(w, q, n)$ simula un paso de computación calculando una nueva configuración a partir de una dada.

Todas las funciones anteriores se construyen empleando recursividad primitiva , por lo que son recursivas. En todas ellas, un valor 0 indica que se ha producido algún error en la ejecución de la máquina de Turing.
Finalmente definimos las funciones:

- $ejec$ encargada de simular la ejecución de un cierto número de pasos de M mediante:
$$ejec(w, q, n, 0) = (w, p, n) \text{ y}$$
$$ejec(w, q, n, t + 1) = paso(ejec(w, q, n, t)).$$

- $momentoparada$ que indica el número de pasos que necesita M para llegar a un estado de parada en una entrada w y se define mediante la expresión
$$momentoparada(w) = \mu t[\pi_2^3(ejec(w, 1, 1, t)) = 0]$$

- f la función que evalúa M mediante
$$f(w) = \pi_1^3(ejec(w, 1, 1, momentoparada(w)))$$

Obviamente, f es un función recursiva parcial lo que demuestra el resultado.
■

Por lo tanto, a resultas del Teorema anterior y de los demostrados en la Sección anterior, podemos concluir que las únicas funciones computables (vía máquinas de Turing) son las recursivas parciales.

Apéndice A

Máquinas RAM

El objetivo de este Apéndice es mostrar, de manera clara, que la definición de algoritmo que hemos tomado en estas notas (máquina de Turing) se corresponde con la informática "real". Para ello, introducimos el modelo de máquinas RAM que, aún siendo abstracciones de los computadores actuales, guardan una gran proximidad con los mismos. Además, demostramos la equivalencia entre este modelo y el de máquinas de Turing.

Los lectores interesados en el tema pueden consultar [11] o [6] para ampliar contenidos.

Sin más preámbulos, veamos la definición formal de las máquinas RAM.

Definición 164 (Máquina RAM) *Una máquina RAM es una dispositivo computacional que consta de los siguientes elementos:*

1. *Una estructura de datos conformada por los siguientes elementos: una sucesión de registros de trabajo $(r_i)_{i\in\mathbb{N}}$ (al registro r_0 le llamaremos acumulador) y una sucesión finita de registros de entrada $(i_1, \ldots, i_n)$.*

2. *Un programa RAM dado como una sucesión finita de instrucciones $\Pi = (\pi_1, \ldots, \pi_m)$, donde cada π_i, con $1 \leq i \leq n$, es una instrucción de alguno de los tipos descritos en las Tablas 1, 2 y 3.*

3. *Un registro especial κ llamado contador de programa.*

Donde los registros de trabajo, los registros de entrada y el contador de programa almacenan números enteros.

De acuerdo a la definición, un programa RAM es una sucesión finita de instrucciones de algunos de los tipos descritos en las Tablas 1, 2 y 3. Tal y como reflejan la mismas, podemos clasificar los tipos de instrucciones

que admite una máquina RAM en tres categorías: instrucciones de acceso a registros, aritméticas y de control de flujo.

Las primeras, recogidas en la Tabla 1, se encargan del trasiego de datos entre los registros de trabajo y los de entrada, así como de la carga de constantes en los mismos. Cabe destacar que las máquinas RAM admiten tres tipos de direccionamiento:

1. Inmediato (denotado por $= j$). En este caso no se produce acceso ni a los registros de trabajo ni a los de entrada, simplemente se carga el valor j prefijado por el programa en el acumulador (nótese que la única instrucción que admite direccionamiento inmediato, dentro de la categoría, es `LOAD`).

2. Directo (denotado por j). Las instrucciones que utilizan este tipo de direccionamiento acceden a los registros por medio de un índice j. Si la instrucción es del tipo `READ` se carga en el acumulador el contenido del registro de entrada j–ésimo, si es del tipo `LOAD`, se carga en el acumulador el contenido del registro de trabajo j–ésimo y si es del `STORE`, se almacena en el registro de trabajo j–ésimo el contenido del acumulador.

3. Indirecto (denotado por $\uparrow j$). En este caso se accede a los registros utilizando como índice el contenido del registro r_j. Obviamente, un direccionamiento del estilo "$\uparrow j$"produciría un error en la ejecución de la máquina RAM, que puede asimilarse a la ejecución de una instrucción `HALT` (descrita más adelante). Las instrucciones que hacen empleo de este direccionamiento son de los tres tipos (`READ`, `STORE` y `LOAD`). En el primer caso se carga el registro de entrada r_j–ésimo en el acumulador, en el segundo se almacena el contenido del acumulador en el registro de trabajo r_j–ésimo y en el tercero, se carga el contenido del registro de trabajo r_j–ésimo en el acumulador.

Las instrucciones de carácter aritmético, recogidas en la Tabla 2, pueden ser de tres tipos:

1. `ADD` (suma): se encarga de sumar al acumulador cierto operando (dicho operando responde a alguno de los másodos de direccionamiento anteriormente explicados).

2. `SUB` (resta): substrae al acumulador un cierto operando (de acuerdo a los mismos másodos de direccionamiento anteriores), y por último

Instrucción	Operando	Semástica
`READ`	j	$r_0 := i_j$
`READ`	$\uparrow j$	$r_0 := i_{r_j}$
`STORE`	j	$r_j := r_0$
`STORE`	$\uparrow j$	$r_{r_j} := r_0$
`LOAD`	$= j$	$r_0 := j$
`LOAD`	j	$r_0 := r_j$
`LOAD`	$\uparrow j$	$r_0 := r_{r_j}$

Tabla 1 *Instrucciones de acceso a los registros*

3. `HALF`: cuyo objetivo es almacenar en el acumulador el cociente de dividir éste por dos. La importancia de esta instrucción reside en que nos permite hacer un desplazamiento a la derecha del acumulador (calcular el cociente de dividir por dos se puede interpretar como un desplazamiento a la derecha en la representación binaria en el que se desprecia el último bit, él correspondiente al guarismo de las "unidades"), de manera económica.

Instrucción	Operando	Semástica
`ADD`	$= j$	$r_0 = r_0 + j$
`ADD`	j	$r_0 = r_0 + r_j$
`ADD`	$\uparrow j$	$r_0 = r_0 + r_{r_j}$
`SUB`	$= j$	$r_0 = r_0 - j$
`SUB`	j	$r_0 = r_0 - r_j$
`SUB`	$\uparrow j$	$r_0 = r_0 - r_{r_j}$
`HALF`		$r_0 := \lfloor \frac{r_0}{2} \rfloor$

Tabla 2 *Instrucciones aritméticas*

Las instrucciones encargadas del control del flujo del programa se recogen en la Tabla 3. Podemos distinguir, *grosso modo*, dos tipos: saltos incondicionales (`JUMP`) o condicionales (`JPOS`, `JZERO` o `JNEG`). Estos últimos atienden a la condición de positividad, negatividad o nulidad del acumulador para llevarse a cabo.

Cabe destacar que el potencial de las máquinas RAM queda inalterado si, por ejemplo, se suprimen las instrucciones `SUB`, `HALF`, `JUMP`, `JPOS`, `JNEG`, puesto que sus tareas se pueden llevar a cabo mediante combinaciones de las restantes. Su inclusión atiende tanto a motivos de claridad como de acercamiento a la sintaxis de los lenguajes de ensamblador de los modernos microprocesadores.

Instrucción	Operando	Semástica
`JUMP`	j	$\kappa := j$
`JPOS`	j	si $r_0 > 0$ entonces $\kappa := j$
`JZERO`	j	si $r_0 = 0$ entonces $\kappa := j$
`JNEG`	j	si $r_0 < 0$ entonces $\kappa := j$
`HALT`		$\kappa := 0$

Tabla 3 *Instrucciones de control de flujo*

Para completar la descripción de una máquina RAM, resta describir cómo se ejecuta un programa. Para ello recurriremos, al igual que en el caso de las máquinas de Turing, al concepto de configuración.

Definición 165 (Configuración de una máquina RAM) *Supongamos dado un programa RAM $\Pi := (\pi_1, \ldots, \pi_m)$ (cada π_i, con $1 \leq i \leq m$, es una instrucción de uno de los tipos descritos en las Tablas 1, 2 y 3) y una sucesión finita de registros de entrada $I := (i_1, \ldots, i_n)$, llamaremos configuración de la máquina RAM a todo par ordenado (κ, R), donde κ es un entero que indica la instrucción a ejecutar y R es una sucesión finita que indica el contenido de los registros de trabajo que en algún momento han sido utilizados.*

Obviamente, a la luz de las instrucciones que admite un máquina RAM, los registros de entrada no se modifican a lo largo de la ejecución del programa RAM, por lo que resultaría superfluo que éstos se reflejaran en la configuración de la máquina.

Como configuración inicial tomamos la dada por el par $(1, \emptyset)$ puesto que, *ad initium*, la instrucción a ejecutar es la primera y los registros de trabajo no han sido inicializados con valor alguno. La trancisión de una configuración a otra se lleva a cabo de acuerdo con la semántica presentada en las Tablas 1, 2 y 3. Obviamente, si la instrucción ejecutada es del tipo `READ`, `STORE`, `LOAD`, `ADD`, `SUB` o `HALF`, el contador de programa se incrementa en 1 si no, este se actualiza de acuerdo a lo indicado por la instrucción.

Un ejemplo sencillo de máquina RAM es el siguiente.

Ejemplo 166 (Multiplicación) *El programa RAM descrito en la Tabla 4 lleva a cabo la multiplicación de dos número enteros, almacenados en i_1 e i_2, empleando el algoritmo de multiplicación binario. El resultado lo almacena en el acumulador, i.e. en el registro r_0.*

Dada la mayor *complejidad*, resulta presumible que su capacidad sea al menos equiparable a la de las máquinas de Turing. El siguiente Teorema formaliza este punto.

Etiqueta	Instrucción	Comentario
1	READ 1	[k indicaría la iteración]
2	STORE 1	[r_1 contiene i_1]
3	STORE 5	[r_5 contiene $i_1 2^k$, actualmente i_1]
4	READ 2	
5	STORE 2	[r_2 contiene i_2]
6	HALF	
7	STORE 3	[r_3 contiene $i_2/2^k$]
8	ADD 3	
9	SUB 2	
10	JZERO 14	
11	LOAD 4	[suma r_5 a r_4 sólo si
12	ADD 5	el k--ésimo bit de i_2 es cero]
13	STORE 4	[r_4 contiene $i_1(i_2 \bmod 2^k)$]
14	LOAD 5	
15	ADD 5	
16	STORE 5	
17	LOAD 3	
18	JZERO 20	[si $\lfloor i_2/2^k \rfloor = 0$ hemos terminado]
19	JUMP 5	[si no se repite]
20	LOAD 4	[el resultado se carga en el acumulador]
21	HALT	

Tabla 4 *Programa RAM para la multiplicación (algoritmo binario).*

Teorema 167 (Simulación de M.T. por máquinas RAM) *Sea $L \subseteq \Sigma^*$ un lenguaje aceptado por una máquina de Turing en tiempo acotado por $f(n)$. Entonces, existe una programa RAM que calcula la función característica de L en tiempo $O(f(n))$.*

Demostración .– Sea $M = (K, \Sigma, \delta, s)$ una máquina de Turing que decide el conjunto L en tiempo $O(f(n))$. Veamos como construir una máquina RAM que decide el mismo lenguajes en tiempo $O(f(n))$.
Suponiendo que $\Sigma := \{\sigma_{-1}, \sigma_0, \ldots, \sigma_n\}$, donde $\sigma_{-1} = \triangleright$ y $\sigma_0 = \sqcup$, podemos identificar los elementos de Σ con los números $-1, 0, \ldots, n$.
Con el fin de emular a la máquina de Turing, el registro r_1 almacenará posición de la unidad de control en la cinta, los registros r_2 y r_3 serán utilizados en las tareas de inicialización del proceso mientras que el resto de los registros estarán destinados a simular la cinta de la máquina de Turing.
El programa RAM comienza copiando la entrada a los registros $r_4, \ldots, r_{n+3}$. Para ello, recorre los registros de entrada hasta que encuentra un 0 (modeliza

al carácter blanco ⊔). La posición en la que se encuentra se almacena en el registro r_2. El fragmento de programa RAM encargado de llevar a cabo esta tarea sería el siguiente:

Etiqueta	Instrucción	Comentario
1	`LOAD =4`	[$r_3 := 4$]
2	`STORE 2`	
3	`LOAD =1`	[$r_2 = 1$]
4	`STORE 3`	
5	`READ ↑2`	[$r_0 := i_{r_2}$]
6	`JZERO 15`	[si $r_0 = 0$, i.e. ⊔, terminamos]
7	`STORE ↑3`	[$r_{r_3} := i_{r_2}$]
8	`LOAD 2`	[incrementamos r_2]
9	`ADD =1`	
10	`STORE 2`	
11	`LOAD 3`	[incrementamos r_3]
12	`ADD =1`	
13	`STORE 3`	
14	`JUMP 5`	[repetimos]
15	...	[hemos terminado de copiar]

Una vez copiada la entrada a los registro pertinentes, el programa RAM simula los movimientos de la máquina de Turing uno a uno. Para ello, asociamos a cada $\sigma_j \in \Sigma$ y $q \in K$, con $\delta(q, \sigma_j) = (q, \sigma_l, D)$, el siguiente fragmento de código:

Etiqueta	Instrucción	Comentario
$N_{q,\sigma}$	`LOAD ↑1`	[leemos el carácter de la cinta]
$N_{q,\sigma} + 1$	`SUB j`	[si es el carácter σ (suponemos que
$N_{q,\sigma} + 2$	`JZERO` $N_{q,\sigma} + 4$	la codificación de σ es j)]
$N_{q,\sigma} + 3$	`JUMP` $N_{q,\sigma+1}$	
$N_{q,\sigma} + 4$	`LOAD` $= l$	[escribimos σ_l en la cinta]
$N_{q,\sigma} + 5$	`STORE ↑1`	
$N_{q,\sigma} + 6$	`LOAD 1`	[nos desplazamos en la cinta]
$N_{q,\sigma} + 7$	`ADD` $= D$	
$N_{q,\sigma} + 8$	`STORE 1`	
$N_{q,\sigma} + 9$	`JUMP` $N_{p,\sigma_{-1}}$	[comenzamos simulación del estado p]

Por último, la aceptación o el rechazo de la entrada pueden simularse mediante una secuencia de instrucciones que carguen 1 ó 0 en el acumulador (el registro r_0) seguidas de una instrucción `HALT`.

Puesto que por cada movimiento de la máquina de Turing se llevan a cabo un número de instrucciones acotado por una constante (dependiente del número de estados y del número de símbolos de la máquina), se necesitan a lo más $O(f(n))$ pasos de la máquina RAM para simularla. ■

El recíproco del anterior resultado es cierto en los siguientes términos:

Teorema 168 (Simulación de maquinas RAM por M.T.) *Sea* $g : D \subseteq \mathbb{N} \longrightarrow \mathbb{N}$ *una función computada por un programa RAM* Π *en tiempo acotado por* $f(n)$*. Entonces, existe una máquina de Turing con siete cintas que computa* $g(N)$ *en tiempo acotado por* $O(f(n)^3)$*.*

Demostración .– Con el fin de demostrar el Teorema, basta describir la máquina de Turing encargada de simular el programa Π.
La utilidad de cada una de las cintas es la siguiente:

- La primera cinta contiene la entrada y nunca se sobrescribe,
- La segunda cinta almacena una representación de los registros de trabajo de la máquina RAM, i.e. de los r's. Dicha representación consiste en una serie de cadenas de la forma $i : r(i)$ (representados en binario) y separadas por ";". Entre ellas admitimos la posibilidad de que existan caracteres blanco, por lo que para indicar el final de la cinta, suponemos que el alfabeto Σ contiene un carácter especial $\lhd$. Cada vez que se actualiza el registro r_i, la anterior cadena $i : r(i)$ se borra, escribiéndose la nueva al final de la cinta (justo antes del $\lhd$) insertando caracteres blancos si es necesario.
- La última cinta está destinada a la salida. Si se alcanza una instrucción `HALT`, se copia el contenido del registro r_0, i.e. de la segunda componente del par $0 : r(0)$, a la última cinta y se para.
- El resto de las cintas son de trabajo.

Para simular el funcionamiento del programa RAM, la máquina de Turing utiliza un conjunto de estados para cada instrucción del programa RAM. ■

En resumen, los teoremas anteriores demuestran la equivalencia computacional de ambos modelos.

Apéndice B

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free"in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of çopyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **ƶou"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **Çover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text

formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent"is called **.Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page"means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **.Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ.ªccording to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History"in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History"section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled .Acknowledgements.or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements.or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History"in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled .Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an .aggregate"if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled .Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License .or any later version.applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliografía

[1] J.G. Brookshear. *Teoría de la computación*. Addison–Wesley Iberoamericana, 1993.

[2] C. Calude. *Theories of Computational Complexity*. North–Holland, 1988.

[3] A. Church. The calculi of lambda–conversion. *Annals of Mathematical Studies*, 6, 1941.

[4] S. A. Cook. The complexity of theorem proving procedures. *ACM Symposium on the Theory of Computation*, pages 151–158, 1971.

[5] R.W. Floyd. New proofs and old theorems in logic and formal linguistics, 1964.

[6] J.E. Hopcroft and J.D. Hullman. *Introduction to Automata theory, Languages and Computation*. Addison–Wesley, 1979.

[7] J.E. Hopcroft and J.D. Motwani, R. Hullman. *Introducción a la teoría de autómatas, lenguajes y computación*. Addison–Wesley, segunda edition, 2001.

[8] S.C. Kleene. General recursive functions of natural numbers. *Mathematische Annalen*, 112:727–742, 1936.

[9] A. Meduna. *Automata and Languages*. Springer–Verlag, 2000.

[10] P. Odifredi. *Classical Recursion Theory*. North–Holland, 1989.

[11] C.H. Papadimitriou. *Computational Complexity*. Addison–Wesley Iberoamericana, 1994.

[12] E. Post. Finite combinatory processes: Formulation i. *J. of Symbolic Logic*, 1:103–105, 1936.

[13] E. Post. A variant of recursively unsolvable problem. *Bull. AMS*, 52:264–268, 1946.

[14] A. Turing. On computable numbers, with an application to entscheidungsproblem. *Proc. London Math. Society*, 42(2):230–265, 1936.

Índice alfabético