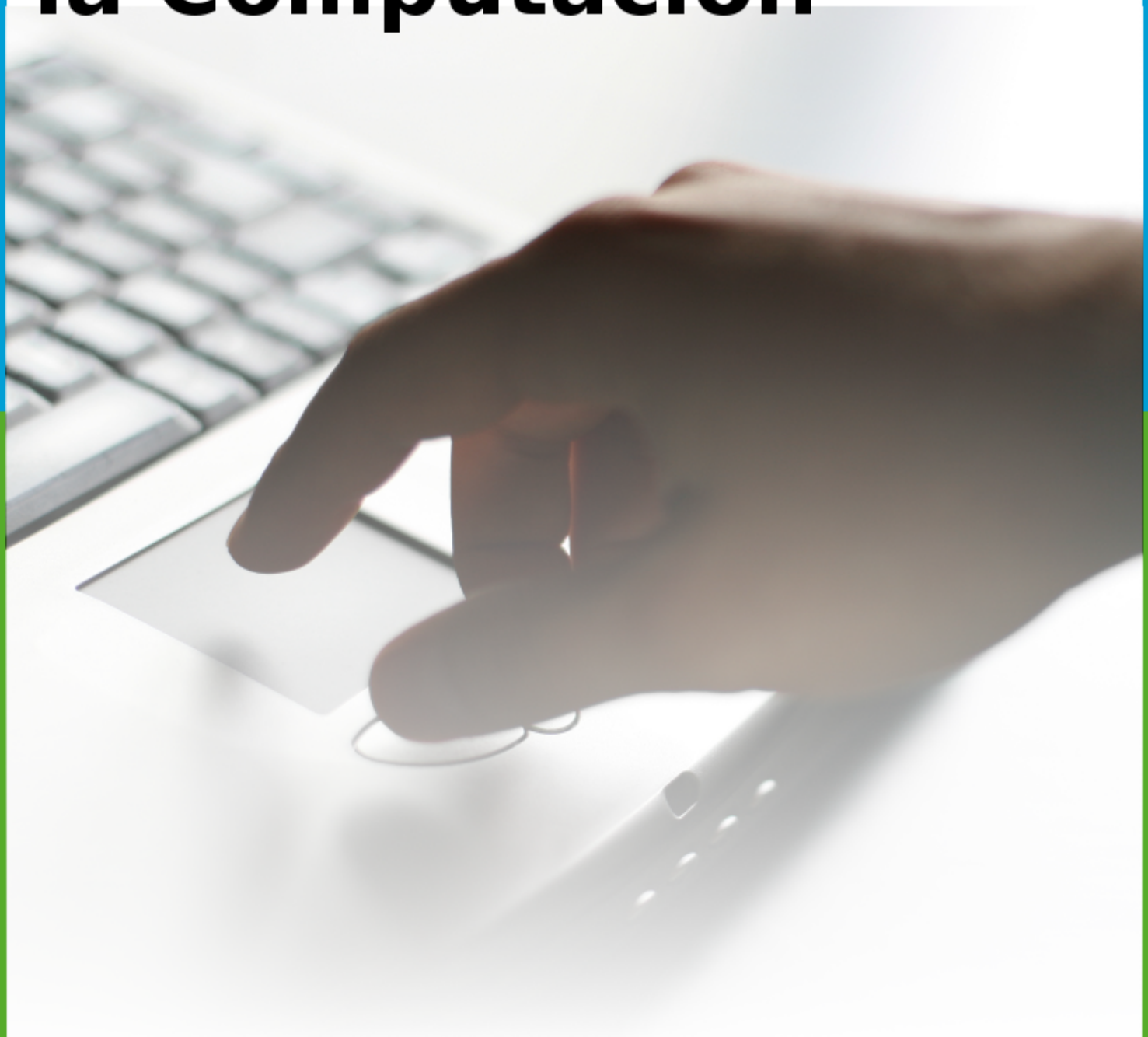
Proyecto LATIn
INICIATIVA LATINOAMERICANA DE LIBROS DE TEXTO ABIERTOS

ALFA
FINANCED BY THE EUROPEAN UNION

Temas de Teoría de la Computación



CC BY SA

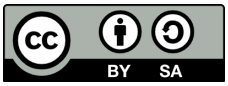
AUTORES

Julio Ariel Hurtado Alegría
Raúl Kantor
Carlos Luna
Luis Sierra
Dante Zanarini

Temas de Teoría de la Computación

1a ed. - Iniciativa Latinoamericana de Libros de Texto Abiertos (LATIn), 2014. 97 pag.

Primera Edición: Marzo 2014
Iniciativa Latinoamericana de Libros de Texto Abiertos (LATIn)
http://www.proyectolatin.org/



Los textos de este libro se distribuyen bajo una licencia Reconocimiento-CompartirIgual 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/deed.es_ES

Esta licencia permite:
Compartir: copiar y redistribuir el material en cualquier medio o formato.
Adaptar: remezclar, transformar y crear a partir del material para cualquier finalidad.

Siempre que se cumplan las siguientes condiciones:

Reconocimiento. Debe reconocer adecuadamente la autoría, proporcionar un enlace a la licencia e indicar si se han realizado cambios. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciador o lo recibe por el uso que hace.

CompartirIgual — Si remezcla, transforma o crea a partir del material, deberá difundir sus contribuciones bajo **la misma licencia que el original**.

Las figuras e ilustraciones que aparecen en el libro son de autoría de los respectivos autores. De aquellas figuras o ilustraciones que no son realizadas por los autores, se coloca la referencia respectiva.

Este texto forma parte de la Iniciativa Latinoamericana de Libros de Texto abiertos (LATIn), proyecto financiado por la Unión Europea en el marco de su Programa ALFA III EuropeAid.
El Proyecto LATIn está conformado por: Escuela Superior Politécnica del Litoral, Ecuador (ESPOL); Universidad Autónoma de Aguascalientes, México (UAA), Universidad Católica de San Pablo, Perú (UCSP); Universidade Presbiteriana Mackenzie, Brasil(UPM); Universidad de la República, Uruguay (UdelaR); Universidad Nacional de Rosario, Argentina(UR); Universidad Central de Venezuela, Venezuela (UCV), Universidad Austral de Chile, Chile (UACH), Universidad del Cauca, Colombia (UNICAUCA), Katholieke Universiteit Leuven, Bélgica (KUL), Universidad de Alcalá, España (UAH), Université Paul Sabatier, Francia (UPS).

Temas de Teoría de la Computación
Libro Colaborativo
Proyecto Latin

Julio Ariel Hurtado Alegria
Universidad del Cauca - Colombia [1]

Raul Kantor
Universidad Nacional de Rosario - Argentina [2]

Carlos Luna
Universidad de la República - Uruguay [3]

Luis Sierra
Universidad de la República - Uruguay [4]

Dante Zanarini
Universidad Nacional de Rosario - Argentina [5]

2014

[1]Capítulo 2 : Complejidad Computacional
[2]Capítulos 3, 4 y 5 : Funciones Recursivas - Funciones de Lista - Máquina de Turing
[3]Ejercicios del Capítulo 1
[4]Capítulo 1 : Lenguajes, Pruebas y Funciones
[5]Ejercicios de los Capítulos 3, 4 y 5

Índice general

1 — Lenguajes, Pruebas y Funciones

Números.

Los números de los que hablamos en este texto son los llamados números naturales: 0, 1, 2, etc. Llamamos $\mathbb{N}$ al conjunto de todos los números. Algunos conjuntos de números se representan convenientemente con las siguientes notaciones:

$$\begin{array}{lcl} [a..b] & := & \{n \in \mathbb{N} : a \leq n \text{ y } n \leq b\} \\ (a..b] & := & \{n \in \mathbb{N} : a < n \text{ y } n \leq b\} \\ [a..b) & := & \{n \in \mathbb{N} : a \leq n \text{ y } n < b\} \\ (a..b) & := & \{n \in \mathbb{N} : a < n \text{ y } n < b\} \end{array}$$

Funciones.

Una relación entre dos conjuntos A y B es un subconjunto del producto cartesiano $A \times B$. Una función de A en B es una relación que además cumple las siguientes condiciones:
totalidad todo elemento de A está relacionado al menos con un elemento de B
funcionalidad todo elemento de A está relacionado a lo más con un elemento de B
Llamamos dominio de la función al conjunto A, y codominio al conjunto B.

Notación usada en las pruebas.

La escritura de las pruebas comprenderá encadenamientos verticales de fragmentos de la siguiente forma:

afirmación 1	afirmación 3	afirmación 5
$\Leftrightarrow$ (justificación A)	$\Leftarrow$ (justificación B)	$\Rightarrow$ (justificación C)
afirmación 2	afirmación 4	afirmación 6

En el primer caso, A es una justificación de que las afirmaciones 1 y 2 dicen precisamente lo mismo; en el segundo caso, B justifica que la afirmación 3 es una consecuencia de la afirmación 4; finalmente, C justifica que la afirmación 5 es una condición suficiente para garantizar la afirmación 6. Por ejemplo, la siguiente es una prueba de que el neutro de la suma es único:

> Supongamos que 0 y $0'$ son dos neutros de la suma. Luego,
>
> $(\forall n :: 0' + n = n)$ y $(\forall m :: m + 0 = m)$
> $\Rightarrow$ (tomando $n = 0$ y $m = 0'$)
> $0' + 0 = 0$ y $0' + 0 = 0'$
> $\Rightarrow$ (transitividad de la igualdad)
> $0 = 0'$.

Otro forma de presentar la misma prueba es la siguiente

$$
\begin{aligned}
&0 = 0' \\
&\Leftarrow \text{ (transitividad de la igualdad)} \\
&0' + 0 = 0 \text{ y } 0' + 0 = 0' \\
&\Leftarrow \text{ (tomando } n = 0 \text{ y } m = 0') \\
&(\forall n :: 0' + n = n) \text{ y } (\forall m :: m + 0 = m),
\end{aligned}
$$

y esto vale cuando 0 y $0'$ son dos neutros de la suma.

1.1 Palabras y lenguajes

Si preguntamos a una persona qué es un lenguaje, posiblemente nos responda por medio de ejemplos: español, inglés, sánscrito. También es posible que nos diga que es una forma en que las personas se comunican naturalmente. Para nosotros, los lenguajes son entidades matemáticas mucho más simples que los lenguajes humanos habituales.

Un lenguaje está compuesto de palabras, y cada palabra se escribe alineando letras en un renglón; las letras están ordenadas, y ese orden es esencial para distinguir las palabras. Por ejemplo, escribimos la palabra "arroz" de la forma siguiente

arroz

Y gracias al orden, que entendemos relevante, distinguimos esa palabra de

zorra

Distinguimos las dos palabras porque, evidentemente, la primera letra de la primera palabra (la 'a') no es la misma que la primera letra de la segunda palabra ('z'). Es decir, podemos distinguir las letras.

Llamamos *alfabeto* a un conjunto de *letras* o *signos* que podemos distinguir. Nuestras primeras palabras, *arroz* y *zorra*, han usado las letras del alfabeto habitual, pero nada nos impide pensar en otros signos. Algunos otros alfabetos usuales son:

$$\{x, y, z\}, \{0,1,2,3,4,5,6,7,8,9\}, \{\alpha, \beta, \gamma\}, \{\bullet, \circ\}.$$

Observemos que los signos elegidos son lo suficientemente claros como para distinguirlos. Evitaremos siempre aquellas definiciones de alfabetos cuyos signos puedan confundirnos. Por ejemplo, si tomamos el alfabeto de las letras del español deberíamos reconocer como letras a los signos "ch" y "ll"; y entonces, la palabra "chanchullo", ¿tiene siete o diez letras? Como no nos interesa entrar en estas disquisiciones, asumimos que nunca podremos confundir los signos, ya sea por separado o en el contexto de una palabra.

De igual manera que escribimos una palabra en español colocando una primera letra, luego una segunda, y así sucesivamente, podemos escribir palabras en estos alfabetos. Por ejemplo, las siguientes son palabras sobre los alfabetos tomados de ejemplo.

$$xyyzz, 8786868, \beta, \bullet\bullet\bullet.$$

Llegamos entonces a nuestra primera

Definición 1.1.1 (Palabra, lenguaje)**.** *Sea* Σ *un alfabeto. Una* palabra *sobre* Σ *es una secuencia finita de letras. Un* lenguaje *sobre* Σ *es un conjunto de palabras sobre* Σ*.*

Para un alfabeto Σ dado, hay un lenguaje muy especial; el de todas las palabras sobre Σ. Llamamos Σ^* a ese lenguaje de todas las palabras sobre Σ.

1.2 Una excursión a las funciones

Hemos afirmado que las palabras son secuencias. Pero, ¿qué es una secuencia? Las siguientes definiciones aclaran estos conceptos en términos matemáticos habituales.

Definición 1.2.1 (Secuencia de largo n, secuencia)**.** *Sean* Σ *un alfabeto y* n *un número. Una* secuencia de largo n *sobre* Σ *es una función con dominio* $[0..n)$ *y codominio* Σ*. Una* secuencia sobre Σ *es una secuencia de largo* m *sobre* Σ *para algún* $m \in \mathbb{N}$.

En este texto usaremos indistintamente los términos palabra y secuencia.
Ahora podemos reescribir parte de nuestra introducción de forma más precisa[1].

Sea el siguiente alfabeto:

$$\Sigma \quad := \quad \{\text{a,b,c,d,e,f,g,h,i,j,k,l,m,n,ñ,o,p,q,r,s,t,u,v,w,x,y,z}\}$$

Definimos las palabras *arroz* y *zorra* como las secuencias de largo 5 tales que

$$\begin{array}{ll} \mathit{arroz}(0) = \text{a} & \mathit{zorra}(0) = \text{z} \\ \mathit{arroz}(1) = \text{r} & \mathit{zorra}(1) = \text{o} \\ \mathit{arroz}(2) = \text{r} & \mathit{zorra}(2) = \text{r} \\ \mathit{arroz}(3) = \text{o} & \mathit{zorra}(3) = \text{r} \\ \mathit{arroz}(4) = \text{z} & \mathit{zorra}(4) = \text{a} \end{array}$$

Como $\mathit{zorra}(0) \neq \mathit{arroz}(0)$, tengo una justificación en términos matemáticos que dichas palabras son distintas.

La lectura cuidadosa de las definiciones puede proporcionar una comprensión mayor de la disciplina que se está estudiando. En nuestro caso, ¿qué significa una secuencia de largo 0? Simplemente una función cuyo dominio es el conjunto vacío. ¿Y cuántas funciones con dominios vacíos existen? Solamente una; porque si F es una función con dominio vacío, debe ser la función vacío.

$$\begin{array}{l} F \subseteq \emptyset \times B \\ \iff \\ F = \emptyset. \end{array}$$

Usamos ε para denotar la palabra vacía[2]. Notemos, además, que una letra y una palabra con una sola letra son cosas muy diferentes: en el segundo caso nos encontramos con una función cuyo dominio es el conjunto unitario $\{0\}$. Sin embargo, no usaremos ninguna notación que nos permita distinguir esto; es decir, • denotará en ocasiones a la letra • y en ocasiones a la palabra cuya única letra es •, dependiendo del contexto.

1.2.1 Concatenación

También podemos expresar en términos matemáticos la concatenación de dos palabras; esto significa escribir una palabra a continuación de la otra. Por ejemplo, de concatenar las palabras *zorra* y *arroz* resulta la palabra *zorraarroz*.

[1] Podríamos decir incluso que de forma algo pedante

[2] Desde el punto de vista de la teoría de conjuntos, que codifica casi todos los conceptos matemáticos como conjuntos, la palabra vacía ε y el conjunto vacío $\emptyset$ son precisamente el mismo conjunto. La confusión desaparece por el uso notacional que hacemos; dependiendo del contexto escribiremos ε o $\emptyset$. Esta situación es exactamente la misma que se presenta con el número cero 0, que también es codificado en la teoría de conjuntos como el conjunto vacío.

Definición 1.2.2 (Concatenación)**.** *La concatenación de las palabras w_0 de largo l_0 sobre Σ_0 y w_1 de largo l_1 sobre Σ_1 es la palabra w de largo $l_0 + l_1$ sobre $\Sigma_0 \cup \Sigma_1$ tal que*

$$w(i) \quad := \quad \text{si } i < l_0 \text{ entonces } w_0(i) \text{ sino } w_1(i - l_0)$$

Habitualmente concatenamos palabras definidas sobre un mismo alfabeto Σ, y no usamos ningún signo para denotar la función de encadenamiento, colocando simplemente los argumentos uno al lado del otro. Es decir, escribimos $w_0 w_1$ para denotar la concatenación de las palabras w_0 y w_1.

La palabra vacía ε es el neutro de la operación de concatenación. En términos algebraicos, decimos que la estructura $\langle \Sigma^*, \text{concatenación}, \varepsilon \rangle$ es un monoide, una estructura algebraica con una operación binaria asociativa con elemento neutro.

1.3 Definiciones de lenguajes

Un lenguaje sobre un alfabeto Σ es un conjunto de palabras, o sea, un subconjunto de Σ^*. En esta sección veremos dos formas muy conocidas de definir subconjuntos: una tercera forma ameritará una nueva sección.

El lenguaje Σ^* es el lenguaje que contiene a todas las palabras sobre el alfabeto Σ. Otro lenguaje muy usado es Σ^+, el lenguaje de todas las palabras no vacías sobre Σ

$$\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$$

Un tercer lenguaje es el lenguaje vacío, que no tiene palabras, y que es precisamente el conjunto vacío $\emptyset$. La palabra vacía se relaciona con estos lenguajes de la forma siguiente:

$$\varepsilon \in \Sigma^* \qquad \varepsilon \notin \Sigma^+ \qquad \varepsilon \notin \emptyset$$

1.3.1 Definiciones por extensión

Podemos definir un lenguaje por extensión, es decir, enumerando todas sus palabras. Por ejemplo, definimos el lenguaje L sobre el alfabeto Σ de la forma siguiente:

$$L \quad := \quad \{\bullet\bullet\bullet\bullet\bullet, \bullet, \bullet\circ, \bullet\bullet\bullet\}$$

Este lenguaje tiene cuatro palabras. Observemos que no hay ninguna regla inmediata que vincule dichas palabras, salvo el haber sido elegidas para integrar el lenguaje L. Una primera limitación de este mecanismo es que solamente nos permite definir lenguajes finitos. Pero sus limitaciones son aún mayores.

Consideremos la siguiente propiedad P: una palabra w sobre Σ cumple con la propiedad P si y solamente si w no es la palabra vacía y la primera de sus letras es $\bullet$. Escribimos esta propiedad como

$$P(w) \quad := \quad w \neq \varepsilon \;\text{ y }\; w(0) = \bullet$$

Consideremos ahora la pregunta siguiente: ¿las palabras del lenguaje L cumplen con la propiedad P? O como haremos habitualmente,

$$(\forall w \in L :: P(w))$$

Como la forma del lenguaje ha sido completamente arbitraria, el único mecanismo que tenemos para comprobar la afirmación anterior es una inspección, palabra por palabra, de cada

elemento del lenguaje, hasta probar la propiedad en cada caso o encontrar un contraejemplo[3]. Para este lenguaje particular debemos realizar solamente cuatro inspecciones, pero es evidente que tendremos dificultades si el lenguaje fuera más grande.

Por otro lado, si pretendo definir una función cuyo dominio sea el lenguaje L, también deberé indicar el valor funcional correspondiente a cada palabra por separado. Una vez más, la tarea se vuelve imposible para tamaños grandes.

1.3.2 Definiciones por comprensión

Podemos definir un lenguaje por comprensión indicando alguna propiedad que deben cumplir todas sus palabras. Por ejemplo, podemos pensar en el siguiente lenguaje L' sobre el alfabeto Σ.

$$L' \quad := \quad \{w \in \Sigma^+ : w(0) = \bullet\}$$

Este mecanismo define subconjuntos de un conjunto dado. En nuestro caso particular, el conjunto dado es Σ^+, y el subconjunto definido es el de las palabras no vacías cuya primera letra es $\bullet$.

Este mecanismo de definición supera la primera limitación que encontramos en la definición por extensión: ya no nos limitamos a los lenguajes finitos. Sin ir más lejos, L' tiene infinitas palabras.

Ahora probemos que las palabras de L' cumplen con la propiedad P de la sección anterior. El esquema de definición nos ayuda proporcionando información acerca de cada palabra del lenguaje. Mostraremos que para probar que la propiedad vale para una palabra w basta mostrar que se encuentra en L'.

$P(w)$
$\Leftarrow$ (Def. P)
$w \neq \varepsilon$ y $w(0) = \bullet$
$\Leftarrow$ (Def. Σ^+)
$w \in \Sigma^+$ y $w(0) = \bullet$
$\Leftarrow$ (Def. por comprensión)
$w \in L'$.

Hemos probado que

$$\big(\forall w \in L' :: P(w)\big)$$

Nuestra prueba tiene éxito gracias al uso de la propiedad que define a nuestro conjunto.

1.4 Definiciones por inducción

En esta sección presentamos un tercer mecanismo de definición de lenguajes, especialmente relevante en lógica e informática. La sintaxis de los lenguajes de programación se definen siguiendo estas ideas.

La forma de esta clase de definiciones proporciona un esquema de prueba y de definición de funciones más simple y poderoso que las formas anteriores. Asímismo, permite la automatización de pruebas y definiciones de funciones sobre lenguajes[4].

Presentaremos dos estilos para definir un lenguaje inductivo. El primer estilo, que llamaremos de reconocimiento, plantea criterios para reconocer las palabras del lenguaje. El segundo estilo, que llamaremos declarativo, enuncia reglas que debe cumplir una familia de lenguajes, siendo el lenguaje definido la intersección de todos ellos. Ambos estilos permiten definir precisamente los mismos lenguajes.

[3] Es decir, una palabra que **no** cumpla con la propiedad.
[4] Y de hecho, esta posibilidad ha sido implementada.

1.4.1 Definiciones inductivas por reconocimiento

La definición inductiva de un lenguaje L sobre un alfabeto Σ puede verse como un proceso infinito de reconocimiento de las palabras de L. Se cuenta con un conjunto de criterios básicos que reconocen un cierto conjunto inicial de palabras de L; luego, se proporcionan criterios de propagación (o inductivos) que muestran cómo el reconocimiento de palabras se va propagando desde palabras ya reconocidas a nuevas palabras por reconocer.

Por ejemplo, para definir el conjunto de las palabras sobre $\Sigma = \{\bullet, \circ\}$ que terminan con $\bullet$, podría escribir

1. **criterio básico** $\bullet \in L''$
2. **criterio de propagació n** $w \in L'' \longrightarrow \bullet w \in L''$
3. **criterio de propagació n** $w \in L'' \longrightarrow \circ w \in L''$

O de forma más compacta,

1. **criterio básico** $\bullet \in L''$
2. **criterio de propagació n** $w \in L''$ y $a \in \Sigma \longrightarrow aw \in L''$

La definición muestra cómo reconocer cada palabra de L''. Con el criterio básico se reconoce la palabra $\bullet$ como palabra del lenguaje. El criterio de propagación permite reconocer una palabra de L'' con la forma aw siempre que conociéramos de antemano que w es una palabra del lenguaje y a una letra del alfabeto.

Cada criterio tiene la forma $\ldots \longrightarrow \ldots$, y lo leemos de la manera siguiente: si se cumplen las condiciones a la izquierda de la flecha, o *precondiciones*, entonces reconocemos las palabras a la derecha de la flecha como pertenecientes al lenguaje que estamos definiendo. En las ocasiones en que no hayan precondiciones, como puede ser el caso de los criterios básicos, escribiremos solamente las palabras del lenguaje que se reconocen siguiendo ese criterio.

Las precondiciones de los criterios pueden hacer referencia o no al conjunto que se está reconociendo: cuando existen esas referencias nos encontramos ante criterios de propagación, y en caso contrario ante criterios básicos. Los criterios de propagación cumplen dos restricciones.

1. Las precondiciones que mencionan al lenguaje que se está reconociendo deben ser de la forma $\ldots \in L$. Es decir, no aceptaremos criterios como el siguiente: $w \notin L \longrightarrow ww \in L$.
2. Las nuevas palabras reconocidas deben contener estrictamente como subpalabras a aquellas referenciadas en la precondición, pero no pueden descomponerlas ni modificarlas. Es decir, no aceptaremos criterios como los siguientes: $wa \in L \longrightarrow aw \in L$ ni $wa \in L \longrightarrow ww \in L$. De esta forma garantizamos que la aplicación de los criterios reconocen palabras cada vez más largas.

En resumen, para definir un lenguaje inductivamente se proporcionan criterios básicos y de propagación; el lenguaje definido es el formado por las palabras reconocidas por esos criterios.

Cada palabra w del lenguaje así definido es reconocida luego de una cadena finita de aplicaciones de los criterios, a la que llamamos *secuencia de formación (o reconocimiento) de w*. Por ejemplo, reconocemos la palabra $\circ\circ\bullet$ con la siguiente secuencia de formación:

1. reconocemos $\bullet$ siguiendo el criterio básico;
2. entonces, aplicamos un criterio de propagación para reconocer la palabra $\circ\bullet$;
3. y terminamos la tarea volviendo a aplicar el mismo criterio de propagación para reconocer la palabra $\circ\circ\bullet$.

Más brevemente, escribiremos

I $\bullet \in L$ (criterio básico 1)
II $\circ\bullet \in L$ (criterio de propagación 3, usando I)
III $\circ\circ\bullet \in L$ (criterio de propagación 3, usando II)

Cada paso de la secuencia de formación se realiza aplicando un criterio básico, o aplicando un criterio de propagación a palabras reconocidas previamente en esa secuencia de formación.

1.4.2 Definiciones inductivas declarativas

Podemos plantearnos otra perspectiva para definir lenguajes inductivamente. Desde este nuevo punto de vista, vemos los criterios como reglas que un lenguaje X sobre Σ pudiera cumplir.

1. **regla básica** $\bullet \in X$
2. **regla inductiva** si $w \in X$ y $a \in \Sigma$ entonces $aw \in X$

Si una familia de lenguajes cumple con las reglas, la intersección de esa familia también cumple con las reglas. El lenguaje L_0 definido inductivamente por las cláusulas anteriores es la intersección[5] de todos los lenguajes X que satisfacen las reglas básicas e inductivas dadas. Acostumbramos escribir la definición escribiendo el nombre del conjunto que se está definiendo en las mismas reglas. En nuestro caso hubiéramos escrito

1. **regla básica** $\bullet \in L_0$
2. **regla inductiva** si $w \in L_0$ y $a \in \Sigma$ entonces $aw \in L_0$

Los criterios básicos del estilo anterior se convierten directamente en reglas básicas de este estilo; los criterios de propagación se traducen inmediatamente como reglas inductivas.

Ambos estilos definen precisamente el mismo lenguaje. Efectivamente, observemos primero que el lenguaje L'' de las palabras reconocidas siguiendo los criterios cumplen inmediatamente con las reglas; basta con reescribir las reglas de la forma siguiente para ponerlo de manifiesto:

1. $\bullet$ es reconocida como palabra por los criterios para L''
2. Si $a \in \Sigma$ y w es reconocida como palabra por los criterios para L'', entonces aw es reconocida como palabra por los criterios para L''

Veamos ahora que L'' es el menor lenguaje que cumple con las reglas para L_0. Sea X un lenguaje que cumple con dichas reglas; mostraremos que todas las palabras reconocidas por los criterios de L'' están en X. Supongamos que hay palabras reconocidas por los criterios para L'' que no están en X, y que w es una de esas palabras con secuencia de formación de largo minimal; es decir, todas las palabras reconocidas con secuencias de formación de largo menor a la que reconoce a w están en X. La palabra w no puede ser $\bullet$, porque como X cumple con la regla básica tenemos $\bullet \in X$. Por lo tanto, w es de la forma aw' donde w' aparece antes en la secuencia de formación de w. Ahora bien, como esa secuencia de formación de w' es de largo menor a la que forma w, tenemos que $w' \in X$; y como X cumple con la regla inductiva, concluímos que $aw' \in X$, contradiciendo nuestro supuesto original. Luego, no hay palabras reconocidas por los criterios para L'' que no estén en X, y por lo tanto $L'' \subseteq X$.

En consecuencia, podemos definir lenguajes inductivos de dos formas diferentes; dando criterios para reconocer palabras del lenguaje, o dando reglas que deben ser satisfechas por los lenguajes.

1.4.3 Ejemplos

El lenguaje Σ^* de todas las palabras sobre el alfabeto Σ se puede definir inductivamente como el lenguaje reconocido por los siguientes criterios:

1. **criterio básico** $\varepsilon \in \Sigma^*$
2. **criterio de propagació n** $w \in \Sigma^*$ y $a \in \Sigma \longrightarrow aw \in \Sigma^*$

O, igualmente, como el menor lenguaje X que cumple las siguientes reglas:

1. **regla básica** $\varepsilon \in X$
2. **regla inductiva** si $w \in X$ y $a \in \Sigma$ entonces $aw \in X$

El resto de los ejemplos de esta sección se presentan con criterios de reconocimiento. El lector puede fácilmente reescribirlos como reglas para el estilo declarativo de definiciones inductivas.

[5] Y por lo tanto el menor, en el sentido de la inclusión, de esos lenguajes.

El lenguaje de las palabras de largo par lo puedo generar a partir de la palabra vacía agregando de a dos letras, como lo muestra la siguiente definición inductiva.

1. **criterio básico** $\varepsilon \in L_{par}$
2. **criterio de propagación** $w \in L_{par}$ y $\{a,b\} \subseteq \Sigma \longrightarrow abw \in L_{par}$

Si partiera de las palabras que tienen una sola letra, estaría generando el lenguaje de las palabras de largo impar.

1. **criterio básico** $a \in \Sigma \longrightarrow a \in L_{impar}$
2. **criterio de propagación** $w \in L_{impar}$ y $\{a,b\} \subseteq \Sigma \longrightarrow abw \in L_{impar}$

Consideremos ahora $\Sigma := \{\bullet, \circ\}$. ¿Cómo podemos definir L_{alt}, el lenguaje de aquellas palabras que no tienen dos letras iguales contiguas?

Una primera respuesta es la siguiente:

1. **criterio básico** $\{\varepsilon, \bullet, \circ\} \subseteq L_{alt}$
2. **criterio de propagación** $\bullet w \in L_{alt} \longrightarrow \circ\bullet w \in L_{alt}$
3. **criterio de propagación** $\circ w \in L_{alt} \longrightarrow \bullet\circ w \in L_{alt}$

Observemos que no estamos violando las restricciones que hemos puesto para las precondiciones, ya que las nuevas palabras reconocidas ($\circ\bullet w$ y $\bullet\circ w$) contienen a la que aparece en la precondición ($\bullet w$ y $\circ w$, respectivamente).

Una alternativa, que involucra otra forma de pensar el problema, es la siguiente: definamos dos lenguajes auxiliares que alternan las letras, pero de forma que conozcamos precisamente con qué letra comienzan. La unión de ambos dará la solución buscada.

1. **criterio básico** $\{\varepsilon, \bullet\} \in L_\bullet$
2. **criterio de propagación** $w \in L_\bullet \longrightarrow \bullet\circ w \in L_\bullet$

1. **criterio básico** $\{\varepsilon, \circ\} \in L_\circ$
2. **criterio de propagación** $w \in L_\circ \longrightarrow \circ\bullet w \in L_\circ$

1. **criterio básico** $L_\bullet \cup L_\circ \subseteq L_{alt}$

En este último caso hemos dado una definición inductiva sin criterios de propagación. Estos casos pueden ser definidos sin necesidad de usar tanta artillería.

$$L_{alt} \quad := \quad L_\bullet \cup L_\circ$$

Otro caso extremo en el uso de las reglas es cuando no hay reglas básicas: en ese caso, estamos definiendo el lenguaje vacío, que no contiene ninguna palabra. Es muy fácil entender esto si lo pensamos desde el punto de vista declarativo.

Supongamos que todas las reglas son de la forma

Si $w \in X$, entonces P

El conjunto vacío cumple con esta regla, ya que no es posible proporcionar un elemento del conjunto vacío que cumpla P. Y como este es el conjunto más pequeño existente, el esquema de definiciones inductivas en su aspecto declarativo nos muestra un mecanismo complicado para definir el conjunto vacío.

El lenguaje vacío $\emptyset$ es un lenguaje muy diferente de $\{\varepsilon\}$, que tiene una única palabra. La forma de definir inductivamente un conjunto finito, indicando uno a uno todos sus elementos, es precisamente el mecanismo de definición por extensión que hemos visto antes.

1.5 Probando propiedades

Es usual que los teoremas tomen la forma

$$(\forall a \in U :: P(a))$$

donde U es un conjunto conocido, y P una propiedad sobre los elementos de dicho conjunto. Un ejemplo muy conocido es el teorema de Pitágoras

> Sea ABC un triángulo rectángulo con hipotenusa AC. Entonces, $|AB|^2 + |BC|^2 = |AC|^2$.

En este caso, U es la clase de los triángulos rectángulos, y $P(ABC)$ es $|AB|^2 + |BC|^2 = |AC|^2$.

En esta sección nos concentraremos en la relación entre la definición de los lenguajes y las pruebas de propiedades sobre sus palabras. En particular veremos que las definiciones inductivas de lenguajes proporcionan una forma de prueba muy útil, la llamada prueba por inducción.

1.5.1 Extensión y comprensión

Supongamos que deseamos probar un teorema

$$(\forall a \in U :: P(a)), \tag{1.1}$$

donde el lenguaje U ha sido definido por extensión como

$$U \quad := \quad \{u_1, u_2, \ldots u_n\}$$

Para probar el teorema 1.1 debemos demostrar que la propiedad vale para cada una de sus palabras.

$$P(u_1) \text{ y } P(u_2) \text{ y } \ldots \text{ y } P(u_n) \qquad \Rightarrow \qquad (\forall a \in U :: P(a))$$

La forma en que definimos el conjunto, donde cada palabra fue tomada arbitrariamente sin ninguna preocupación acerca de su relación con las restantes, no nos proporciona ninguna ayuda acerca de cómo probar una propiedad para todas las palabras de un lenguaje. Para cada palabra del lenguaje debemos probar la propiedad P independientemente de las restantes palabras del mismo lenguaje. El mecanismo de definir un conjunto por extensión no nos proporciona ninguna estrategia interesante para probar propiedades.

La situación cambia un poco con la definición de un conjunto por comprensión. Si nuestro universo se hubiera definido como

$$U \quad := \quad \{a \in A : Q(a)\},$$

lo que precisamos para probar el teorema es mostrar que

$$(\forall a \in A :: Q(a) \Rightarrow P(a))$$

Para encontrar una prueba del teorema 1.1 debemos relacionar las propiedades P y Q. Supongamos que estamos estudiando el lenguaje de las palabras no vacías sobre el alfabeto $\{\circ, \bullet\}$ que no tienen $\bullet$, y la propiedad de que todas las palabras comienzan con la letra $\circ$:

$$\begin{aligned} U \quad &:= \quad \{w \in \{\circ, \bullet\}^+ : w \text{ no tiene } \bullet\} \\ P(w) \quad &:= \quad w \text{ empieza con } \circ \end{aligned}$$

El siguiente argumento prueba que toda palabra $w \in \{\circ, \bullet\}^+$ que no tenga $\bullet$ (o sea, que cumpla $Q(w)$), debe empezar con $\circ$ (o sea, debe cumplir $P(w)$).

> Sea $w \in \{\circ, \bullet\}^+$; por lo tanto $w \neq \varepsilon$, y w tiene una primera letra. Como w cumple con la propiedad Q, esa letra no puede ser $\bullet$. Por lo tanto, esa letra es $\circ$, lo que muestra que w cumple la propiedad P.

En la sección siguiente veremos que en el caso de las definiciones inductivas, el mecanismo de definición colabora en el mecanismo de la prueba proporcionando además nueva información.

1.5.2 Lenguajes inductivos

Definamos inductivamente $L'_\circ$ de la forma siguiendo.

1. **regla básica** $\circ \in L'_\circ$
2. **regla inductiva** si $w \in L'_\circ$ entonces $\circ\bullet w \in L'_\circ$

Para probar una propiedad P sobre todas las palabras del lenguaje, nos bastará con probar los siguientes lemas

1. (**caso base**) $P(\circ)$
2. (**caso inductivo**) Si $P(w)$, entonces $P(\circ\bullet w)$

Por ejemplo, demostremos que cada palabra $w \in L'_\circ$ cumple la propiedad $P(w)$ que dice: "la cantidad de letras $\circ$ que aparecen en w es una más que la cantidad de $\bullet$". Para el caso base debo justificar que $P(\circ)$; este resultado es inmediato, porque la cantidad de letras $\circ$ que aparece es una, y la cantidad de letras $\bullet$ que aparece es cero. Para el caso inductivo debo justificar que $P(\circ\bullet w)$, y puedo usar como hipótesis adicional (hipótesis inductiva) que $P(w)$; como en w hay una letra $\circ$ más que $\bullet$, y en $\circ\bullet w$ agrego tanto una $\circ$ como una $\bullet$, seguiré teniendo una letra $\circ$ más.

Esta prueba sigue el principio de inducción primitiva que probaremos a continuación para el lenguaje $L'_\circ$. Las condiciones que se plantean en la hipótesis siguen las reglas que definen a $L'_\circ$.

Teorema 1.5.1 (Principio de inducción primitiva para $L'_\circ$)**.** *Sea P una propiedad sobre $L'_\circ$. Si se cumplen las siguientes condiciones*

1. *$P(\circ)$*
2. *Dado $w' \in L'_\circ$, si $P(w')$ entonces $P(\circ\bullet w')$*

entonces

$$\big(\forall w \in L'_\circ :: P(w)\big)$$

Demostración. Defino el conjunto X de todas las palabras de $L'_\circ$ que cumplen con la propiedad P.

$$X \quad := \quad \big\{w \in L'_\circ : P(w)\big\}$$

Vamos a probar que X cumple con las reglas de la definición de $L'_\circ$.

1. (**regla básica**) Debemos probar que $\circ \in X$. Esto es inmediato por la condicion 1 del teorema.
2. (**regla inductiva**) Debemos probar que si $w \in X$ entonces $\circ\bullet w \in X$.

$$\begin{array}{l} w \in X \\ \Rightarrow \text{(Def. } X\text{)} \\ P(w) \\ \Rightarrow \text{(condicion 2 del teorema)} \\ P(\circ\bullet w) \\ \Rightarrow \text{(Def. } X\text{)} \\ \circ\bullet w \in X. \end{array}$$

El conjunto X satisface las reglas que definen al lenguaje $L'_\circ$. Como este lenguaje es el mínimo conjunto que cumple dichas reglas, tenemos

$$L'_\circ \subseteq X.$$

Como todas las palabras de $L'_\circ$ están en X, concluímos

$$\big(\forall w \in L'_\circ :: P(w)\big).$$

■

El principio de inducción primitiva para $L'_\circ$ (o PIP para $L'_\circ$) proporciona un mecanismo para probar que todas las palabras de $L'_\circ$ satisfacen una propiedad. En este mecanismo, las reglas inductivas proporcionan hipótesis adicionales llamadas *hipótesis inductivas*.

Mostramos ahora el uso explícito de este principio en la prueba de la propiedad que hablaba de la cantidad de letras $\bullet$ y $\circ$ que aparecen en las palabras de $L'_\circ$.

Teorema 1.5.2 (Cantidad de letras de las palabras de $L'_\circ$)**.** *Considere la siguiente propiedad P sobre palabras de $L'_\circ$:*

$$P(w) \quad := \quad \textit{la cantidad de letras } \circ \textit{ que aparecen en } w \textit{ es una más que la cantidad de } \bullet.$$

Luego, $(\forall w \in L'_\circ :: P(w))$

Demostración. Vamos a probar las condiciones del PIP para $L'_\circ$ y la propiedad P.

Caso base. Voy a probar que $P(\circ)$ se cumple. Inmediato, porque la cantidad de letras $\circ$ que aparece es una, y la cantidad de letras $\bullet$ que aparece es cero.

Caso inductivo. Voy a probar que si $P(w)$ se cumple, entonces también se cumple $P(\circ\bullet w)$. Como en w hay una letra $\circ$ más que $\bullet$ (hipótesis inductiva), y en $\circ\bullet w$ agrego tanto una $\circ$ como una $\bullet$, entonces la cantidad de $\circ$ que hay en $\circ\bullet w$ sigue siendo una más que la cantidad de $\bullet$.

Aplicando el PIP para $L'_\circ$ y la propiedad P, concluyo que todas las palabras de $L'_\circ$ cumplen la propiedad deseada. ■

Cada definición inductiva de un lenguaje L proporciona un teorema, el PIP para L, que se prueba de forma análoga al PIP para $L'_\circ$. Llamamos pruebas inductivas a aquellas que aplican el PIP para algún lenguaje; al escribirlas debemos explicitar el lenguaje inductivo y la propiedad considerados, al igual que todos los casos bases e inductivos que sean necesarios.

1.6 Más sobre definiciones inductivas

1.6.1 Relaciones inductivas

Podemos extender las ideas de la definición inductiva de conjuntos para definir relaciones inductivamente. La idea es que las relaciones definidas sean subconjuntos de $A \times B$, donde A es un lenguaje inductivo y B un conjunto arbitrario. Para ello indicamos el universo en que se encuentra la relación a definir, y luego se establecen las reglas correspondientes, asegurándose que las primeras coordenadas sean una copia de la definición inductiva de A. Por ejemplo, podemos definir inductivamente la relación $R_0 \subseteq L'' \times \Sigma^*$ con las siguientes reglas:

1. **regla básica** si $w \in \Sigma^*$ entonces $\langle \bullet, \bullet w \rangle \in R_0$
2. **regla inductiva** si $a \in \Sigma$ y $\langle w, w' \rangle \in R_0$ entonces $\langle aw, aw' \rangle \in R_0$

Usando el PIP para L'' probaremos fácilmente que R_0 relaciona cada palabra de L'' con alguna palabra de Σ^*.

Teorema 1.6.1 (Totalidad de R_0)**.** *Considere la siguiente propiedad P sobre palabras de L'':*

$$P(w) \quad := \quad \left(\exists w' \in \Sigma^* :: \langle w, w' \rangle \in R_0\right).$$

Luego, $(\forall w \in L'' :: P(w))$.

Demostración. Vamos a probar las condiciones del PIP para L'' y la propiedad P.

Caso base. Voy a probar que $P(\bullet)$ se cumple. Inmediato, porque la regla básica de la definición de R_0 garantiza que $\langle \bullet, \bullet \rangle \in R_0$.

Caso inductivo. Voy a probar que si $a \in \Sigma$ y $P(w)$ se cumple, entonces también se cumple $P(aw)$. La hipótesis inductiva garantiza la existencia de $w' \in \Sigma^*$ tal que $\langle w, w' \rangle \in R_0$; la regla inductiva de la definición de R_0 nos permite afirmar que $\langle aw, aw' \rangle \in R_0$.

Aplicando el PIP para L'' y la propiedad P, concluyo que todas las palabras de L'' aparecen como primer componente en la relación R_0. ■

1.6.2 Varias definiciones de un mismo lenguaje

En esta sección mostramos que un mismo lenguaje puede tener más de una definición inductiva. Consideremos el alfabeto $\Sigma = \{\bullet, \circ\}$, y los lenguajes L_1 y L_2 definidos inductivamente por las siguientes familias de reglas.

1. **regla básica** $\varepsilon \in L_1$
2. **regla básica** $\bullet \in L_1$
3. **regla básica** $\bullet\circ \in L_1$
4. **regla inductiva** si $\{w, w'\} \subseteq L_1$ entonces $ww' \in L_1$

1. **regla básica** $\varepsilon \in L_2$
2. **regla inductiva** si $w \in L_2$ entonces $\bullet w \in L_2$
3. **regla inductiva** si $w \in L_2$ entonces $\bullet\circ w \in L_2$

Mostraremos que L_1 y L_2 son el mismo lenguaje; dos definiciones diferentes para un mismo objeto matemático. La prueba de esta igualdad aparece en los siguientes lemas y teoremas. Observe que una vez probada la igualdad de los conjuntos, tenemos dos PIP que nos permiten probar propiedades sobre las palabras del mismo lenguaje.

Teorema 1.6.2 ($L_2 \subseteq L_1$)**.** *El lenguaje L_2 está incluído en L_1.*

Demostración. Probaremos usando el PIP para L_2 la propiedad P siguiente:

$$P(w) \quad := \quad w \in L_1$$

Caso base 1. Hay que probar $P(\varepsilon)$. Inmediato, por la regla básica 1 para L_1.

Caso inductivo 2. Hay que probar $P(\bullet w)$. La hipótesis inductiva nos permite afirmar $P(w)$, es decir, $w \in L_1$. Además, la regla básica 2 para L_1 garantiza que $\bullet \in L_1$. Finalmente, la regla inductiva 4 para L_1 justifica que $\bullet w \in L_1$.

Caso inductivo 3. Se deja al lector.

Aplicando el PIP para L_2, se concluye que todas las palabras de L_2 están en L_1. ■

La inclusión restante se prueba fácilmente usando el siguiente lema.

Lema 1.6.1. *El lenguaje L_2 es cerrado bajo la operación de concatenación.*

Demostración. Probaremos usando el PIP para L_2 la propiedad P siguiente:

$$P(w) \quad := \quad \big(\forall w' \in L_2 :: ww' \in L_2\big)$$

Caso base 1. Hay que probar $P(\varepsilon)$. Inmediato, ya que la propiedad se reduce a probar que $w' \in L_2$ suponiendo que $w' \in L_2$.

Caso inductivo 2. Hay que probar $P(\bullet w)$. Tomemos una palabra $w' \in L_2$. La hipótesis inductiva nos permite afirmar $P(w)$, y por lo tanto $ww' \in L_2$. Finalmente, la regla inductiva 2 para L_2 justifica que $\bullet ww' \in L_2$.

Caso inductivo 3. Se deja al lector.
Aplicando el PIP para L_2, se concluye que L_2 es cerrado bajo la operación de concatenación. ■

Teorema 1.6.3 ($L_1 \subseteq L_2$)**.** *El lenguaje L_1 está incluído en L_2.*

Demostración. Probaremos usando el PIP para L_1 la propiedad P siguiente:

$$P(w) \quad := \quad w \in L_2$$

Caso base 1. Hay que probar $P(\varepsilon)$. Inmediato, por la regla básica 1 para L_2.
Caso base 2. Hay que probar $P(\bullet)$. La regla básica L_2 garantiza que $\varepsilon \in L_2$. Finalmente, la regla inductiva 2 para L_2 justifica que $\bullet \in L_2$.
Caso base 3. Se deja al lector.
Caso inductivo 4. Hay que probar $P(ww')$. Las hipótesis inductivas nos permiten afirmar $P(w)$ y $P(w')$, es decir, $w \in L_2$ y $w' \in L_2$. Finalmente, el lema justifica que $ww' \in L_2$.
Aplicando el PIP para L_1, se concluye que todas las palabras de L_1 están en L_2. ■

1.6.3 Definiciones inductivas libres

Una palabra de un lenguaje inductivo puede ser reconocida de múltiples maneras. Decimos que una definición inductiva es libre si las múltiples formas de reconocer una misma palabra terminan en idéntica aplicación del mismo criterio de reconocimiento. La propiedad de ser libre no es una propiedad de un lenguaje, sino de una definición inductiva del mismo.

Consideremos las definiciones L_1 y L_2 presentadas en la sección anterior, y las relaciones $R_1 \subseteq L_1 \times \Sigma^*$ y $R_2 \subseteq L_2 \times \Sigma^*$ definidas inductivamente con las siguientes reglas.

1. **regla básica** $\langle \varepsilon, \varepsilon \rangle \in R_1$
2. **regla básica** $\langle \bullet, \bullet \rangle \in R_1$
3. **regla básica** $\langle \bullet\circ, \bullet\circ \rangle \in R_1$
4. **regla inductiva** si $\{\langle w_1, w \rangle, \langle w_2, w' \rangle\} \subseteq R_1$ entonces $\langle w_1 w_2, w_2 \rangle \in R_1$

1. **regla básica** $\langle \varepsilon, \varepsilon \rangle \in R_2$
2. **regla inductiva** si $\langle w, w' \rangle \in R_2$ entonces $\langle \bullet w, \bullet w' \rangle \in R_2$
3. **regla inductiva** si $\langle w, w' \rangle \in R_2$ entonces $\langle \bullet\circ w, \bullet w' \rangle \in R_2$

Podemos probar fácilmente, al igual que en el teorema 1.6.1, que R_1 y R_2 cumplen la propiedad de totalidad. Pero además podemos probar que la relación R_2 cumple la propiedad de funcionalidad, mientras que la de R_1 no. La propiedad de funcionalidad se desprende de que la definición de L_2 es libre.

Primero mostramos que R_1 no es funcional.

Teorema 1.6.4 (R_1 no es funcional)**.** *Existen dos palabras w y w' de Σ^* y una palabra $u \in L_1$ tales que*

$$\langle u, w \rangle \in R_1 \text{ y } \langle u, w' \rangle \in R_1 \text{ y } w \neq w'$$

Demostración. Tomemos

$$u := \bullet\bullet\bullet \text{ y } w := \bullet \text{ y } w' := \bullet\bullet.$$

Evidentemente, $w \neq w'$. La siguiente secuencia de formación muestra que $\langle u, w \rangle \in R_1$.

I $\langle \bullet, \bullet \rangle \in R_1$ (regla básica 2)
II $\langle \bullet\bullet, \bullet \rangle \in R_1$ (criterio de propagación 4, usando I las dos veces)
III $\langle \bullet\bullet\bullet, \bullet \rangle \in R_1$ (criterio de propagación 4, usando I y II, con $w_1 = \bullet\bullet$ y $w_2 = \bullet$)
Se deja al lector el dar una secuencia de formación para $\langle u, w' \rangle$ y terminar la prueba. ■

Antes de mostrar que la definición de L_2 es libre es conveniente probar el siguiente lema.

Lema 1.6.2. *Ninguna palabra de L_2 comienza con $\circ$.*

Demostración. Se deja al lector. ■

Ahora mostraremos que

Teorema 1.6.5. *La definición de L_2 es libre.*

Demostración. Usaremos un análisis de casos[6] siguiendo la definición de L_2. La propiedad a probar para $w \in L_2$ es que toda forma de reconocer w termina en una idéntica aplicación del mismo criterio de reconocimiento.

Caso base 1. Observamos que la única forma de terminar el reconocimiento de ε es aplicando el criterio básico.

Caso inductivo 2. Consideremos que hay dos formas esencialmente distintas de finalizar el reconocimiento de la palabra $\bullet w \in L_2$. Una de ellas debe resultar de aplicar el criterio 2, y la restante de aplicar el criterio 3. Por lo tanto, la palabra w debe ser de la forma $\circ w'$ y pertenecer a L''. Pero esto contradice el lema.

Caso inductivo 3. Se deja al lector.

■

Teorema 1.6.6 (Funcionalidad de R_2)**.** *La relación R_2 es una relación funcional.*

Demostración. Considere la siguiente propiedad P sobre palabras de L_2:

$$P(u) \quad := \quad \left(\forall \left\{w, w'\right\} \subseteq \Sigma^* :: \langle u, w \rangle \in R_2 \text{ y } \left\langle u, w' \right\rangle \in R_2 \Rightarrow w = w'\right)$$

Para probar la funcionalidad demostraremos que todas las palabras de L_2 cumplen con la propiedad P.

Caso base 1. Voy a probar que $P(\varepsilon)$ se cumple. Como $\langle \varepsilon, w \rangle \in R_2$, y el último criterio para reconocer $\varepsilon \in L_2$ es el criterio básico, concluímos que $w = \varepsilon$. De igual manera, $w' = \varepsilon$.

Caso inductivo 2. Considero dos palabras w y w' de Σ^* tales que $\langle \bullet u, w \rangle \in R_2$ y $\langle \bullet u, w' \rangle \in R_2$. Como L_2 es libre, el último criterio para reconocer $\bullet u$ debió ser el criterio de propagación 2; y por lo tanto hay dos palabras w_H y w'_H de Σ^* tales que $\langle u, w_H \rangle \in R_2$, $\langle u, w'_H \rangle \in R_2$, y además $w = \bullet w_H$ y $w' = \bullet w'_H$. La hipótesis inductiva garantiza que $w_H = w'_H$, y por lo tanto $w = w'$.

Caso inductivo 3. Se deja al lector.

La aplicación del PIP para L_2 termina la prueba. ■

1.7 Definiciones de funciones

Hemos visto tres formas de definir lenguajes: por extensión, por comprensión, y por inducción. También hemos visto cómo probar propiedades sobre lenguajes definidos de esa manera; en particular, vimos el Principio de Inducción Primitivo para los lenguajes definidos inductivamente. En esta sección veremos cómo definir funciones cuyos dominios son lenguajes definidos inductivamente.

[6] La prueba por análisis de casos presenta la misma estructura que la correspondiente a la aplicación del PIP. Se distingue de la prueba inductiva porque no hace uso de ninguna hipótesis inductiva.

1.7.1 Contar •: dominios definidos por extensión o comprensión

Vamos a definir funciones que cuentan la cantidad de • en lenguajes definidos de distinta manera. Intentemos ver cómo vincular la forma de definir el lenguaje se vincula con la forma de definir la función.

Comencemos mirando el lenguaje definido por extensión

$$L = \{\bullet\bullet\bullet\bullet\bullet, \bullet, \bullet\circ, \bullet\bullet\bullet\}$$

Para definir la función con dominio L que indica la cantidad de • proporcionamos un valor funcional a cada palabra de L. Por ejemplo, podemos definir $\#_\bullet^L$ de la forma siguiente:

$$\begin{aligned}
&\#_\bullet^L : L \to \mathbb{N} \\
&\#_\bullet^L(\bullet\bullet\bullet\bullet\bullet) := 5 \\
&\#_\bullet^L(\bullet) \quad\quad := 1 \\
&\#_\bullet^L(\bullet\circ) \quad := 1 \\
&\#_\bullet^L(\bullet\bullet\bullet) \quad := 3
\end{aligned}$$

O también mostrando la función como un subconjunto de $L \times \mathbb{N}$:

$$\#_\bullet^L \quad := \quad \{\langle \bullet\bullet\bullet\bullet\bullet, 5\rangle, \langle \bullet, 1\rangle, \langle \bullet\circ, 1\rangle, \langle \bullet\bullet\bullet, 3\rangle\}$$

Para definir una función cuyo dominio sea un lenguaje definido por extensión, debemos indicar el valor funcional de cada palabra sin tomar en cuenta ninguna propiedad adicional, ya que el mecanismo de definición no provee propiedades adicionales.

La situación puede empeorar cuando los dominios están definidos por comprensión. Por ejemplo, cuando el dominio es el lenguaje definido por comprensión:

$$L' = \{w \in \Sigma^+ : w(0) = \bullet\}$$

la definición de L' no proporciona ninguna ayuda para definir la función $\#_\bullet^{L'}$.

1.7.2 Contar •: dominios definidos inductivamente

Ahora veamos la situación cuando tomamos un lenguaje definido inductivamente, por ejemplo el lenguaje L_2 de la página 18. Recordemos que una función es solamente una relación que cumple las propiedades de totalidad y funcionalidad. La idea es definir la función $\#_\bullet^{L_2}$ como una relación inductiva R en el universo $L_2 \times \mathbb{N}$. Las reglas que definen R son:

1. **regla básica** $\langle \varepsilon, 0\rangle \in R$
2. **regla inductiva** si $\langle w, n\rangle \in R$ entonces $\langle \bullet w, 1+n\rangle \in R$
3. **regla inductiva** si $\langle w, n\rangle \in R$ entonces $\langle \bullet\circ w, 1+n\rangle \in R$

Al igual que lo hicimos con la relación R_2 podemos probar fácilmente que R satisface las propiedades de totalidad y funcionalidad, y es por lo tanto una función legítimamente definida.

Teorema 1.7.1. *R es una función con dominio L_2.*

Demostración. Se deja al lector. ■

Podemos demostrar que R es una función gracias a que consideramos una definición libre para el dominio L_2. En general, a partir de una definición inductiva libre de un lenguaje A podemos definir una función $f : A \to B$ transformando la definición de A en la definición de una relación inductiva en $A \times B$. Esta forma de definir una función es llamada Esquema de Recursión Primitivo (ERP) para esa definición de A.

En vez de definir la relación inductiva R mediante reglas, acostumbramos darla en términos de ecuaciones. Este estilo de presentación es solamente una reescritura para la definición de la relación inductiva R. Por ejemplo, la función $\#_\bullet^{L_2}$ que cuenta las letras $\bullet$ de las palabras de L_2 la escribimos de la forma siguiente:

$$\begin{aligned}
&\#_\bullet^{L_2} : L_2 \to \mathbb{N} \\
&\#_\bullet^{L_2}(\varepsilon) \quad = 0 \\
&\#_\bullet^{L_2}(\bullet w) \;\; = 1 + \#_\bullet^{L_2}(w) \\
&\#_\bullet^{L_2}(\bullet \circ w) = 1 + \#_\bullet^{L_2}(w)
\end{aligned}$$

Este estilo de definiciones proporciona casi inmediatamente un programa que computa la función deseada. Por ejemplo, si contáramos con un lenguaje de programación suficientemente astuto, podríamos codificarla de la forma siguiente.

```
FUNCTION cant (w:L2): nat;
BEGIN
  CASE w OF
    ε : cant := 0; break;
    •w': cant := 1 + cant (w'); break;
    •∘w': cant := 1 + cant (w'); break;
  END CASE;
END;
```

Al definir inductivamente un lenguaje estamos dando elementos que nos permiten construír un algoritmo para el cómputo de funciones. Con esto queremos decir que los lenguajes inductivos proporcionan, gracias a la estructura de su definición, elementos que colaboran en la construcción de programas.

El esquema de definición recursiva nos indica que para definir una función f debemos proporcionar una regla de cómputo a cada regla de definición del dominio. Las reglas de cómputo asociadas a las reglas básicas usan aquellos elementos que aparecen en las precondiciones de la regla. Cuando no hay precondiciones, la regla de cómputo simplemente asigna un valor constante a la función. En el ejemplo de $\#_\bullet^{L_2}$ teníamos

$$\#_\bullet^{L_2}(\varepsilon) = 0$$

Las reglas inductivas presentan un cómputo ligeramente más complejo. Recordemos una de las reglas de la definición de R y su correspondiente ecuación:

$\#_\bullet^{L_2}(\bullet \circ w) = 1 + \#_\bullet^{L_2}(w)$
3 **regla inductiva** si $\langle w, n\rangle \in R$ entonces $\langle \bullet \circ w, 1+n\rangle \in R$

La segunda coordenada w' en la precondición de la regla es el valor funcional $\#_\bullet^{L_2}(w)$; en programación usamos el nombre *llamada o invocación recursiva* a ese cómputo.

1.8 Ejercicios

Ejercicio 1.8.1. Dado el alfabeto $\Sigma = \{a, b, c\}$, defina inductivamente los siguientes lenguajes sobre Σ:

1. $\{\varepsilon, ab, abab, ababab, abababab, ...\}$.
2. $\{ab, aabb, aaabbb, aaaabbbb, ...\}$.
3. Las palabras que empiezan con a.
4. Las palabras que terminan con c.

Ejercicio 1.8.2. Dado un alfabeto Σ, sobre el cual está definido un orden total, defina inductivamente los siguientes lenguajes:

1. Las palabras ordenadas. Una palabra está ordenada si cada letra de la misma, excepto la última, es menor o igual que la que le sigue.
2. Las palabras que son capicúas.

Ejercicio 1.8.3. Dado el alfabeto $\Sigma = \{a,b\}$, se pide:

1. Proponga dos definiciones inductivas diferentes del lenguaje de todas las posibles palabras sobre Σ.
2. Pruebe la igualdad de los conjuntos inductivos anteriores.

Ejercicio 1.8.4. Dado un alfabeto Σ, defina las siguientes funciones:

1. *largo*, que dada una palabra w de Σ^*, retorna la cantidad de letras que tiene w.
2. *pertenece*, que dada una palabra w de Σ^* y una letra x de Σ, retorna $true$ si x está en w, y $false$ en caso contrario.
3. *elim*, que dadas una palabra w de Σ^* y una letra x de Σ, retorna la palabra w sin (eventualmente) la primera ocurrencia de x.
4. *elimTodas*, que dada una palabra w de Σ^* y una letra x de Σ, retorna la palabra w sin (eventualmente) ninguna ocurrencia de x.

Ejercicio 1.8.5. Pruebe, por indución, que para toda palabra w de Σ^* y toda letra x de Σ, se cumplen:

1. $largo(elim(w,x)) \leq largo(w)$.
2. $largo(elimTodas(w,x)) \leq largo(elim(w,x))$.
3. $pertenece(elimTodas(w,x),x) = false$.

Ejercicio 1.8.6. Dado un alfabeto Σ, sobre el cual está definido un orden total, defina las siguientes funciones:

1. *ordenada*, que dada una palabra w de Σ^*, retorna true si, y sólo si, w está ordenada de menor a mayor.
2. *insOrd*, que dada una palabra w de Σ^*, ordenada de menor a mayor, y dada una letra x de Σ, retorna la palabra ordenada (de menor a mayor) compuesta por todas las letras de w y por x.
3. *ord*, que dada una palabra w de Σ^*, retorna la palabra ordenada (de menor a mayor) compuesta por todas las letras de w.
4. Pruebe, para toda palabra w de Σ^*: $ordenada(ord(w)) = true$.

Ejercicio 1.8.7. Defina inductivamente las siguientes relaciones binarias entre palabras de un alfabeto Σ:

1. $prefijo$, de las palabras que son prefijos de otras. Por ejemplo, *amo* es prefijo de: *amor*, *amores* y *amo*, entre otras.
2. $posfijo$, de las palabras que son posfijos de otras. Por ejemplo, *la* es posfijo de *hola*.

2 — Complejidad Computacional

2.1 Algoritmos

Definición 2.1.1 (Algoritmo)**.** *Secuencia determinista de pasos que ejecutados por una máquina real permite resolver una instancia del problema.*

Un algoritmo resuelve problemas computacionales para los cuales éste provee una salida de validación.
Las propiedades de un algoritmo son:

- Precisión (en cada instrucción)
- Determinismo (en cada instrucción)
- Finitud (en el conjunto finito de instrucciones)

El algoritmo ejecuta sus instrucciones y se detiene si no hay más instrucciones que ejecutar.

Definición 2.1.2 (Computabilidad)**.** *Es la capacidad de convertir las entradas en salidas por medio de una relación (*input *x* output*). Define qué puede y qué no ser computable.*

Definición 2.1.3 (Complejidad)**.** *Tiempo necesario para ejecutar un algoritmo que resuelve un problema.*

Definición 2.1.4 (Corrección)**.** *Un algoritmo es* CORRECTO *si* $\forall$ input *válido luego de comenzar la ejecución del algoritmo, éste se detiene y produce un* output *correcto para el* input.

Un algoritmo es *INCORRECTO* si $\exists$ *input* válido para el cual:

1. El algoritmo no se detiene
2. Calcula un *output* incorrecto

2.2 Corrección de los Algoritmos (Por Inducción)

Considere que se quiere construir un algoritmo que para una secuencia $S = (s_1 s_2 ... s_n)$ y un natural $n \geqslant 1$ que representa el largo de la secuencia, permita determinar el mayor elemento de la secuencia. Para ello se ha definido la función $MAX(S,n)$., veamos como dos soluciones una iterativa y otra recursiva se analizan para determinar si están correctamente implementadas

2.2.1 Corrección de Algoritmos Iterativos

Este tipo de algoritmos presenta el problema de los *loops*, que para otros casos resulta siendo trivial. Tales *loops* tienen una *corrección parcial* si el algoritmo se detiene, entonces entrega un valor correcto. Se define la terminación cuando el algoritmo no tiene más instrucciones por ejecutar.

Una solución iterativa para la función $MAX(S,n)$ es:
Para este caso:

Algorithm 1 $MAX(S,n)$

```
1: m ← s_1
2: k ← 2
3: while k ≤ n do
4:     if s_k > m then
5:         m ← s_k
6:     end if
7:     k ← k + 1
8: end while
9: return m
```

- Se *d*etiene porque k se incrementa en 1 en cada iteración hasta que se alcanza $n+1$, en cuyo caso se sale del ciclo $WHILE$ y el programa termina
- Correctitud (Inducción)

Hay que buscar una invariante en los ciclos, una expresión lógica que sea verdadera antes de iniciar el ciclo y al final de cada iteración del ciclo → expresión que une las variables.
Si la expresión es verdadera siempre implica que se mantiene al finalizar el ciclo.
Es necesario elegir la expresión que más información nos entregue respecto a lo que queremos que el algoritmo haga.

Lema 2.2.1. *Si $MAX(S,n)$ se ejecuta con una secuencia S de valores $s_1,s_2,...,s_n$, entonces retorna el máximo de la secuencia.*

Debemos encontrar una invariante del *loop*, una expresión que siempre se mantenga verdadera.

Al final $m_n \geq s_k \ \forall k \leq n-1; k \in \mathbb{N}; m_n \geq s_1,s_2,...,s_n$
(*) Durante $m_i \geq s_k \ \forall k \leq n-1; m \in S \ y \ m \geq s_1,s_2,...,s_{k-1}; k \in \mathbb{N}; m_i \geq s_1,s_2,...,s_{i-1}$ (*i-esima iteración*)
(BI) Para $i=0$; $m_i = m_0 = s_1$ y $k_i = k_0 = 2$ (antes de comenzar)
(HI) Al final de cada iteración $im_i \geq s_k \forall k \leq i-1$
(TI) Al final de cada iteración $i+1$ el valor de k se incrementa en 1. $k_{i+1} = k_i + 1$
Ahora el valor de m se modifica dependiendo del IF de las líneas 4, 5 y 6, dependiendo de la comparación con s_k. Los valores que se comparan $m_i \ y \ s_{k_i}$ son los de la iteración anterior (i en nuestro caso).
Tenemos dos casos:

1. $m_i + 1 = m_i$ si $m_i \geq Sk_i$
2. $m_i + 1 = Sk_i$ si $m_i < Sk_i$

Dividamos la demostración:

1. Si $m_i \geq Sk_i$, entonces al final de la iteración sabremos que $m_i + 1 = m_i$ y $k_{i+1} = k_i + 1$
   Por *(HI)* sabemos que $m_i > s_1,s_2,...,Sk_{i-1}$ y como $m_i \geq Sk_i \Rightarrow m_i > s_1,s_2,s_3,...,Sk_i$.
   Como $m_{i+1} = m_i$, entonces $m_{i+1} > s_1,s_2,s_3,...,Sk_i$.

   Pero $k_{i+1} = k_i + 1$
   $k_i = k_{i+1} - 1$
   $m_i + 1 > s_1,s_2,s_3,...,Sk_{i+1} - 1$
2. Si $Sk_i > m_i$, entonces al final de la iteración $i+1$ sabremos que $m_{i+1} = Sk_i$ y $k_{i+1} = k_i + 1$
   Por *(HI)* sabemos que $m_i \geq s_1,s_2,s_3,...,Sk_i - 1$
   Dado que $Sk_i > m_i$ obtenemos que $Sk_i \geq s_1,s_2,s_3,...,Sk_i - 1$
   En este caso $m_{i+1} = Sk_i$

$m_{i+1} \geq s_1, s_2, s_3, ..., Sk_{i-1}$
Dado $k_i + 1$; $k_i = ki + 1 - 1$
$m_{i+1} \geq s_1, s_2, s_3, ..., Sk_{i+1} - 1$
$\Rightarrow$ la propiedad se cumple para $i+1$
Como *(*)* es la invariante de $MAX(S,n)$ y siempre se cumple que $m \in S$ y $m \geq s_1, s_2, ..., s_{k-1}$.
Si el algoritmo termina en $k = n+1 \Rightarrow m \geq s_1, s_2, ..., s_n$; $m \in S$ y $m \geq s_1, s_2, ..., s_n$ y por tanto *m es el máximo*.
$MAX(S,n)$ calcula correctamente el máximo.

2.2.2 Corrección de Algoritmos Recursivos

En este caso no se hace división entre terminación y corrección parcial, por consiguiente se demuestra la propiedad deseada por inducción.

Sea "A con tamaño $i \Rightarrow$ correcto". Demostrar que "A con tamaño $i+1$ también será correcto".
"Por *curso de valores*, si se ejecuta un *input* de tamaño menor que $i \Rightarrow$ es correcto", a partir de esto se define que para el tamaño i también será correcto.

Una implementación de la función recursiva MAXR es:
Solución Recursiva.

Algorithm 2 $MAXR(S,n)$

```
1:  if n = 1 then
2:      return s_1
3:  else
4:      m ← MAXR(S, n − 1)
5:      if s_n ≥ m then
6:          return s_n
7:      else
8:          return m
9:      end if
10: end if
```

Teorema 2.2.1. *$MAXR(S,n)$ es correcta, es decir retorna el máximo de la secuencia $s_1, s_2, ..., s_n$ de largo n.*

✓ La demostración la haremos sobre el largo
✓ Demostraremos que para todo valor de i se cumple que el resultado retornado por $MAXR(S,i)$ es el máximo de los valores de $s_1, s_2, ..., s_i$; $MAXR(S,i) \geq s_1, s_2, ..., s_i$
(BI) $i = 1$. $MAXR(S,1)$; $s_1 \geq s_1$ *(V)*
(HI) Si se ejecuta $MAXR(S,i)$ el resultado será el máximo de los valores $s_1, s_2, ..., s_n, s_i$
(TI) Queremos demostrar que si se ejecuta $MAXR(S,i+1)$, el resultado será el máximo de los valores
Si se ejecuta $MAXR(S,i+1)$ no debemos preocuparnos sino de las líneas 4 a 10, ya que $i \geq 1$ y por consiguiente $i+1 > 1$
Línea 4 $\rightarrow MAXR(S,i)$ $(i = (i+1)-1)$, por *(HI)* es *(V)*
$\Rightarrow m$ será el máximo valor de $s_1, s_2, ..., s_i$
Ahora tenemos dos posibilidades que se ejecute las líneas 6 u 8.
Si $s_{i+1} \geq m \Rightarrow s_i + 1 \geq s_1, s_2, s_3, ..., s_i$ y por tanto $s_{i+1} \geq s_1, s_2, s_3, ..., s_i, s_{i+1}$
$\Rightarrow MAXR(S,i+1)$ entregaría el resultado correcto s_{i+1}

Si $m > s_{i+1} \Rightarrow m \geq s_1, s_2, s_3, ..., s_i, s_{i+1}$
$\Rightarrow MAXR(S, i+1)$ entregaría el resultado correcto m
Por inducción $\forall i$ se cumple que el resultado referenciado por $MAXR(S,i)$ es el máximo de los valores $\geq s_1, s_2, ..., s_i$ y por tanto $MAXR(S,i)$ es correcto.

2.3 Notación Asintótica

Correcto o no. En la práctica un algoritmo puede ser correcto pero el tiempo de ejecución puede ser demasiado alto.

Definición 2.3.1 (Complejidad)**.** *Estimación del tiempo que tomaría un algoritmo para ejecutarse en función del tamaño de la entrada (lo más independiente del lenguaje).*

Figura 2.1: Ciclo de Análisis y Diseño de Algoritmos.

Para destacar:

- No nos interesa el "*tiempo exacto*" sino el "*orden de crecimiento*" del tiempo necesario respecto al tamaño de la entrada
- Trataremos con funciones del dominio de los $\mathbb{N}$ y recorridos de los reales positivos $\mathbb{R}^+$

Definición 2.3.2. *Sean $f : \mathbb{N} \to \mathbb{R}^+$ y $g : \mathbb{N} \to \mathbb{R}^+$ funciones, definimos $O(g(n))$ como el conjunto de todas las funciones f tales que $\exists c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ que cumple: $\forall n > n_0$, $f(n) \leq cg(n)$. ($g(n)$ es una cota superior definida)*
Cuando $f(n) \in O(g(n))$ diremos que $f(n)$ es a lo más de orden $g(n)$ o simplemente $f(n)$ es $O(g(n))$.
Definimos $\Omega(g(n))$ como el conjunto de todas las funciones f tales que $\exists c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ que cumple $\forall n > n_0$, $f(n) \geq cg(n)$
Cuando $f(n) \in \Omega(g(n))$ diremos que $f(n)$ es a lo menos de orden $g(n)$ o simplemente $f(n)$ es $\Omega(g(n))$.
Finalmente, definimos $\Theta(g(n))$ como la intersección entre $O(g(n))$ y $\Omega(g(n))$, es decir el conjunto de todas las funciones f tales que $f(n) \in O(g(n))$ y $f(n) \in \Omega(g(n))$.
Definiendo de manera similar a lo anterior, decimos que $\Theta(g(n))$ es el conjunto de todas las funciones f tales que $\exists c_1, c_2 \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ que cumplen que $\forall n > n_0$, $f(n) \leq c_1 g(n)$ y $f(n) \geq c_2 g(n)$

Cuando $f(n) \in \Theta(g(n))$ diremos que $f(n)$ es (exactamente) de orden $g(n)$ o simplemente $f(n)$ es $\Theta(g(n))$.

Ejemplo 2.3.1. *Dada la función $f(n) = 10n^2$ es $\Theta(n^2)$*

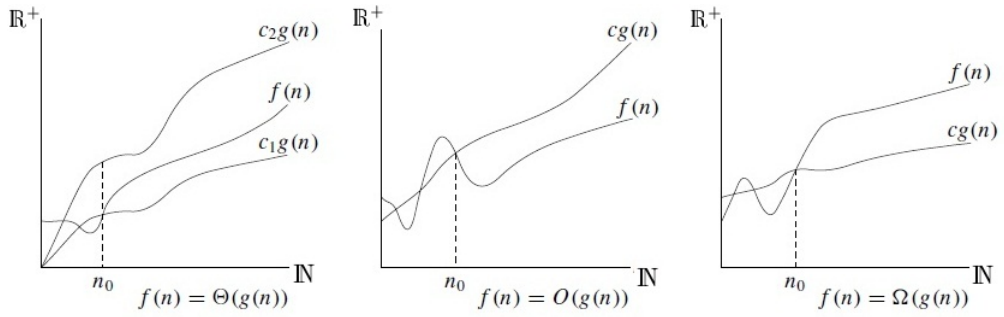


Figura 2.2: Representación Gráfica Notación Asintótica.

1. *Demostrar que* $f(n)$ *es* $O(n_2)$
   Debemos encontrar c *y* n_0 *tal que* $\forall n > n_0$ *se cumpla* $f(n) \leq cn^2$.
   En caso que $n_0 = 0$ *y* $c = 10\ldots$
   $\forall n > 0$, $f(n) = 10n^2 \leq 10n^2$
   $\therefore f(n)$ *es* $O(n^2)$
2. *Demostrar que* $f(n)$ *es* $\Theta(n^2)$.
   Encontrar un c *y* n_0 *tal que* $\forall n > n_0$ *se cumpla que* $f(n) \geq cn^2$.
   Con los mismos valores $n_0 = 0$ *y* $c = 10$, *tenemos que...*
   $\forall n > 0$, $f(n) = 10n^2 \leq 10n^2$
   $\therefore$ *se cumple que* $f(n)$ *es* $\Omega(n^2)$

Finalmente concluimos que $f(n) = 10n^2$ *es efectivamente* $\Theta(n^2)$

Ejemplo 2.3.2. *Dada la función* $f(n) = 60n^2 + 5n + 1$ *es* $\Theta(n^2)$

1. *Demostrar que* $60n^2 + 5n + 1$ *es* $O(n)$
   Si $n = 1$ *y* $c = 66$, *tenemos que...*
   $\forall n \geq 1$, $f(n) = 60n^2 + 5n + 1 \leq 60n^2 + 5n^2 + 1n^2 \leq 66n^2$
   $\therefore f(n)$ *es orden* $O(n^2)$
2. $\forall n \geq 0$, $f(n) = 60n^2 + 5n + 1 \geq 60n^2$
   Si $c = 60$ $n_0 = 0$, *obtenemos que* $f(n)$ *es orden* $\Omega(n^2)$
   $\therefore f(n) = 60n^2 + 5n + 1$ *es* $\Theta(n^2)$

Conclusión

- Las constantes no influyen en la notación Θ
- Sólo el término con el mayor exponente influye en la notación Θ

Teorema 2.3.1. *Si* $f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_2 n^2 + a_1 n + a_0$, *con* $a_i \in \mathbb{R}$ *y tal que* $a_k > 0$, *entonces se cumple que* $f(n)$ *es* $\Theta(n^k)$.

- Primer caso
  $f(n) \leq | a_k | n^k + | a_{k-1} | n^k + \ldots + | a_0 | n^k$
  $f(n) \leq (\sum_{i=0}^{k} a_i) nk$
  $c = \sum_{i=0}^{k} a_i$
  $\Rightarrow f(n)$ es $O(n^k)$
  $n_0 \geq 1$

- Segundo caso

$f(n) \geq a_k n^k - \sum_{i=0}^{k-1} a_i)$; para $n \geq 0$
$c = a_k$

$n_0 = \sum_{i=0}^{k-1} a_i$
$\therefore f(n) \geq a_k n^k$, para $n_0 = \sum_{i=0}^{k-1} a_i$

Ejemplo 2.3.3. *La función* $f(n) = \log_2(n)$ *es* $\Theta(\log_3(n))$
Supongamos que $\log_2(n) = x$ *y que* $\log_3(n) = y$*, de esto obtendremos que* $2x = 3y$
$x \log_2 2 = y \log_2 3$
$x = y \log_2 3$
$log_2(n) = \log_2(3) \log_3(n)$
Resulta que:
$\forall n > 1$, $\log_2(n) \leq \log_2(3) \log_3(n)$
$\forall n > 1$, $\log_2(n) \geq \log_2(3) \log_3(n)$
Si usamos $c = \log_2(3)$ *y* $n_0 = 1$*, resulta que* $t(n) = \log_2(n)$.
Se cumple que es $O(\log_3(n))$ *y* $\Omega(\log_3(n))$ *simultáneamente y por tanto es* $\Theta(\log_3(n))$

Teorema 2.3.2. *Si* $f(n) = \log_a(n)$ *con* $a > 1$*, entonces para todo* $b > 1$ *se cumple que* $f(n)$ *es* $\Theta(\log_b(n))$.

Esto implica que nos podemos independizar de la base del logaritmo. Así en la función logarítmica solo hablamos de $\Theta(\log n)$, sin usar base. Si se requiere de algún cálculo, generalmente supondremos que la base es 2.

Ejemplo 2.3.4. *Mostrar que la función* $f(n) = \sqrt{n}$ *es* $O(n)$ *pero no es* $\Omega(n)$ *(y por tanto no es* $\Theta(n)$*).*
$\forall n \geq 0 \qquad \sqrt{n} \leq n$
Con $c = 1$ *y* $n = 0$*,* $f(n) = \sqrt{n}$ *es* $O(n)$
Ahora mostraremos que $f(n) = \sqrt{n}$ *no es* $\Omega(n)$*, lo haremos por contradición.*
$\exists c > 0$ *y* $n_0 \in \mathbb{N}$*, tal que*
$\forall n \geq n_0 \qquad \sqrt{n} \geq cn$
Concluimos
$\forall n \geq n_0 \qquad \sqrt{n} \geq cn^2$
$\Rightarrow n \geq c^2 n^2$
$\Rightarrow 1 \geq c^2 n$
$\Rightarrow c^2 \leq 1/n$
$\Rightarrow c^2 \leq \lim_{n \to 0} \frac{1}{n}$
$\Rightarrow c^2 \leq 0 \qquad c \notin \mathbb{R}^+$
$\therefore \sqrt{n}$ *no es* $\Omega(n)$

A continuación se presentan las distintas funciones que pertenecen a distintos órdenes (Θ, O, Ω)

Notación	Nombre
$\Theta(1)$	Constante
$\Theta(\log n)$	Logarítmico
$\Theta(n)$	Lineal
$\Theta(n\log n)$	$n\log n$
$\Theta(n^2)$	Cuadrático
$\Theta(n^3)$	Cúbico
$\Theta(n^m)$	Polinomial
$\Theta(k^n)$	Exponencial
$\Theta(n!)$	Factorial

Con m y k constantes positivas $m \geq 0$ y $k \geq 2$

2.4 Complejidad de Algoritmos Iterativos

El tiempo de ejecutar en función del tamaño, lo notaremos $T(n)$ (en notación asintótica).
Estimar: Contar las instrucciones (a veces una en particular y obtener para ese valor una notación asintótica).

Ejemplo 2.4.1. *Consideremos el siguiente trozo de código:*

Algorithm 3

```
1: for i ← 1 to n do
2:     for j ← 1 to i do
3:         x ← x + 1
4:     end for
5: end for
```

El número de veces que se ejecuta la instrucción (3):
$(1)+(1+2)+(1+2+3)+(1+2+3+4)+\ldots$
1 vez *2 veces* *3 veces* *4 veces* $\ldots$
$T(n) = 1+2+3+\ldots+n = \frac{n(n+1)}{2}$
De donde concluimos que
$T(n)$ *es* $\Theta(n^2)$

Ejemplo 2.4.2. *Consideremos el siguiente fragmento de código:*

Algorithm 4

```
1: j ← n
2: while j ≥ 1 do
3:     for i ← 1 to j do
4:         x ← x + 1
5:     end for
6:     j ← ⌊j/2⌋
7: end while
```

Si while *se ejecuta k-veces:*
$T(n) = n + \frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^{k-1}}$
$T(n) = \sum_{i\leftarrow 0}^{k-1} \frac{n}{2^i}$

$\Rightarrow$ *Por serie geométrica tenemos*
$\sum_{i\leftarrow 0}^{k-1} \frac{n}{2^i} = n\sum_{i\leftarrow 0}^{k-1}(\frac{1}{2^i}) = n\frac{1-(\frac{1}{2})^k}{1-(\frac{1}{2})} = 2n(1-(\frac{1}{2})^k)$
Dado que $1-(\frac{1}{2})^k \leq 1$ *obtenemos que* $\forall n > 1$, $T(n) = \sum_{i\leftarrow 0}^{k-1} \frac{n}{2i} \leq 2n$ *y por tanto* $T(n)$ *es* $O(n)$.
$\Rightarrow$ *Por otro lado, como inicialmente para la línea (4) con* $j = n$ *se ejecuta n veces, concluimos que (al menos eso ejecuta):*
$T(n) \geq n$ *y por tanto* $T(n)$ *es* $\Omega(n)$.
$\therefore$ *La cantidad de veces que se ejecuta la línea (4) es* $\Theta(n)$.
En los ejemplos anteriores: tomar en cuenta las instrucciones representativas y contar cuantas veces se repite. El algoritmo 5 busca k en la serie S de tamaño n.

Algorithm 5 Buscar(S,n,k)

```
1: for i ← 1 to n do
2:     if S_i = k then
3:         return i        el índice donde aparece
4:     end if
5: end for
6: return 0
```

Línea representativa $S_i = k$

$\left\{\right.$	*Peor caso*	*k no aparece*	$\Theta(n)$
$\left.\right\}$	*Mejor caso*	*k aparece de primero*	$\Theta(1)$

Entonces Busca(S,n,k) *es* $\Theta(n)$ *en el peor caso y* $\Theta(1)$ *en el mejor caso.*

Ejemplo 2.4.3 (Algoritmo InsertSort)**.** *El algoritmo 6 muestra el reconocido algoritmo de ordenamiento de una secuencia S de tamaño n a través de la inserción incremental de los elementos de tal forma que siempre se respeta el orden.*

Algorithm 6 insertSort(S,n)

```
1: for i ← 2 to n do
2:     t ← S_i
3:     j ← i − 1
4:     while S_j > t ∧ j ≥ 1 do
5:         S_{j+1} ← S_j
6:         j ← j − 1
7:     end while
8:     S_j ← t
9: end for
```

Para la línea (4) se presentará el mejor caso *siempre que* $S_j \leq t$
$\underbrace{2\ldots n}_{n-1 \text{ veces}} \Rightarrow \Theta(n)$

Entre tanto, el peor caso *se presenta si* $j < 1$*, es decir:*
1 vez + 2 veces + 3 veces+...+(n-1)=$\frac{n(n-1)}{2}$
De modo que el peor caso *es* $\Theta(n^2)$.

ejemplo
El algoritmo 7 muestra un proceso para establecer si un número es o no primo.

Algorithm 7 esPrimo(n)
Require: Un número $n \in \mathbb{N}$
Ensure: **true** si n es *primo* y **false** en caso contrario.
1: **for** $i \leftarrow 2$ to $n-1$ **do**
2: **if** $n \bmod i = 0$ **then**
3: **return false**
4: **end if**
5: **end for**
6: **return true**

Al medir la línea (2) determinanos que:
El Mejor caso *se presenta si n es par* $\Rightarrow \Theta(1)$
El Peor caso *se presenta si n es primo* $\Rightarrow \Theta(n)$
Pero el valor numérico n no es una buena estimación, lo que conduce a cambiar $n-1$ *por* $\sqrt{n}$*, de forma que el* peor caso *corresponde a* $\Theta(\sqrt{n})$

En algoritmos numéricos una mejor estimación es d, la cantidad de dígitos necesarios para representar a n.
$d \approx \log_{10}(n)$; $n \approx 10^d \Rightarrow O(k^d)$ (Orden exponencial)
Dado el tamaño d y tomando $\sqrt{n}$ $(n^{\frac{1}{2}})$ en lugar de n, se tiene que $10^{\frac{d}{2}} \geq 3^d$, lo que conduce a $\Omega(k^d)$. Sigue siendo exponencial.

2.5 Propiedades

2.5.1 Propiedades de O

1. $\forall f, f \in O(f)$
2. $f \in O(g) \Rightarrow O(f) \subset O(g)$
3. $O(f) = O(g) \Leftrightarrow f \in O(g)$ y $g \in O(f)$
4. Si $f \in O(g)$ y $g \in O(h) \Rightarrow f \in O(h)$
5. Si $f \in O(g)$ y $f \in O(h) \Rightarrow f \in (min(g,h))$
6. (Regla de la Suma) Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 + f_2 \in O(max(g,h))$
7. (Regla del Producto) Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 * f_2 \in O(g \cdot h)$
8. Si $\exists \lim_{n\to\infty} \frac{f(n)}{g(n)} = k$, dependiendo de k tenemos:
   a) Si $k \neq 0$ y $k < \infty, O(f) = O(g)$
   b) Si $k = 0$ entonces $f \in O(g)$, es decir $O(f) \subset O(g)$, sin embargo se verifica que $g \notin O(f)$

2.5.2 Propiedades de Ω

1. $\forall f$ se tiene que $f \in \Omega(f)$
2. $f \in \Omega(g) \Rightarrow \Omega(f) \subset \Omega(g)$
3. $\Omega(f) = \Omega(g) \Leftrightarrow f \in \Omega(g)$ y $g \in \Omega(f)$
4. Si $f \in \Omega(g)$ y $g \in \Omega(h) \Rightarrow f \in \Omega(h)$
5. Si $f \in \Omega(g)$ y $f \in \Omega(h) \Rightarrow f \in (max(g,h))$
6. (Regla de la Suma) Si $f_1 \in \Omega(g)$ y $f_2 \in \Omega(h) \Rightarrow f_1 + f_2 \in \Omega(min(g,h))$
7. (Regla del Producto) Si $f_1 \in \Omega(g)$ y $f_2 \in \Omega(h) \Rightarrow f_1 * f_2 \in \Omega(g \cdot h)$
8. Si $\exists \lim_{n\to\infty} \frac{f(n)}{g(n)} = k$, dependiendo de k obtenemos:

a) Si $k \neq 0$ y $k < \infty, \Omega(f) = \Omega(g)$
b) Si $k = 0$ entonces $g \in \Omega(f)$, es decir $\Omega(g) \subset \Omega(f)$, sin embargo se verifica que $f \notin \Omega(g)$

2.5.3 Propiedades de Θ

1. $\forall f$ se tiene que $f \in \Theta(f)$
2. $f \in \Theta(g) \Rightarrow \Theta(f) \subset \Theta(g)$
3. $\Theta(f) = \Theta(g) \Leftrightarrow f \in \Theta(g)$ y $g \in \Theta(f)$
4. Si $f \in \Theta(g)$ y $g \in \Theta(h) \Rightarrow f \in \Theta(h)$
5. Si $f \in \Omega(g)$ y $f \in \Omega(h) \Rightarrow f \in (max(g,h))$
6. (Regla de la Suma) Si $f_1 \in \Theta(g)$ y $f_2 \in \Theta(h) \Rightarrow f_1 + f_2 \in \Theta(max(g,h))$
7. (Regla del Producto) Si $f_1 \in \Theta(g)$ y $f_2 \in \Theta(h) \Rightarrow f_1 * f_2 \in \Theta(g \cdot h)$
8. Si $\exists \lim_{n\to\infty} \frac{f(n)}{g(n)} = k$, dependiendo de k tenemos:
   a) Si $k \neq 0$ y $k < \infty, \Theta(f) = \Theta(g)$
   b) Si $k = 0$ entonces los órdenes de f y g son distintos.

2.5.4 Notación o

Definición 2.5.1. $o(g(n)) = \{f(n) : \forall c > 0 \exists n_0 > 0$ *tal que* $0 \leq f(n) < cg(n)$ *para todo* $n > n_0\}$
$f(n)$ *es relativamente insignificante a* $g(n)$ *si* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

La notación o es bien ajustada. Por ejemplo, $2n^2 \in o(n^2)$ es más ajustada que $2n \in O(n^2)$ (O no es asintóticamente estricto)

2.5.5 Notación ω

Análogamente, para Ω se define ω (en forma más ajustada):

Definición 2.5.2. $\omega(g(n)) = \{f(n)$ *para alguna constante* c *se tiene que* $\exists n_0 > 0$ *tal que* $0 \leq cg(n) < f(n)$ *para todo* $n > n_0\}$
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Ejemplo. Caso promedio del método insertSort Analicemos el método *insertSort*
$\underbrace{1 \ldots j-1}_{j/2}$ para insertar S_j
$(n-1)(n/2) = \Theta(n^2)$ Dado el análisis del peor de los casos.
Análisis Probabilístico

$(n-1) \sum_{j\leftarrow 1}^{n} \frac{j}{n}$
$\frac{(n-1)}{n} \sum_{j\leftarrow 1}^{n} j$
$\frac{(n-1)}{n} n \frac{n-1}{2}$
$\frac{(n-1)^2}{2}$
$\therefore \Theta(n^2)$

2.6 Complejidad de Algoritmos Recursivos

2.6.1 Enfoque Divide y Vencerás

Muchos algoritmos son recursivos en sus estructuras, de modo que al resolver un problema ellos se llaman recursivamente. Estos algoritmos exhiben las siguientes componentes:
Dividir $\rightarrow$ Fragmentar el problema en subproblemas
Vencer $\rightarrow$ Resolver los problemas recursivamente
Combinar $\rightarrow$ Coloca las soluciones a los subproblemas para hacer la solución del problema original

Ejemplo 2.6.1 (Algoritmo MergeSort)**.** *Características de MergeSort:*
Dividir $\rightarrow$ *Secuencia* S_n *que será ordenada en dos secuencias de* $n/2$ *elementos cada una*
Vencer $\rightarrow$ *Ordenar las dos secuencias recursivamente usando merge-sort*
Combinar $\rightarrow$ *"Merger" las dos subsecuencias ordenadas para producir la respuesta ordenada*
Fin de recursión $\rightarrow n = 1$*, devuelve un elemento.*

$$\Theta(n), recorrer = \begin{cases} Clave & (merge) \\ merge(A,p,q,r) & p \leq q < r \\ A_{p-q}\ y\ A_{q+1-r} & están\ ordenados. \end{cases}$$

Algorithm 8 merge(A,p,q,r)

```
1:  n1 ← q − p + 1
2:  n2 ← r − q
3:  crear arreglos L[1, n1 + 1] y R[1, n2 + 1]
4:  for i ← 1 to n1 do
5:      L[i] ← A[p + i − 1]
6:  end for
7:  for i ← 1 to n2 do
8:      R[j] ← A[q + j]
9:  end for
10: L[n1 + 1] ← ∞
11: L[n2 + 1] ← ∞
12: i ← 1
13: j ← 1
14: for k ← p to r do
15:     if L[i] ≤ R[j] then
16:         A[k] ← L[i]
17:         i ← i + 1
18:     else
19:         A[k] ← R[j]
20:         j ← j + 1
21:     end if
22: end for
```

Lo anterior generaría dos arreglos $A[p,q]$ *y* $A[q,r]$ *respectivamente.*

Algorithm 9 merge-sort(A, p, r)

```
1: if p < r then
2:     q ← ⌊(p+r)/2⌋
3:     merge-sort(A, p, r)
4:     merge-sort(A, q+1, r)
5:     merge(A, p, q, r)
6: end if
```

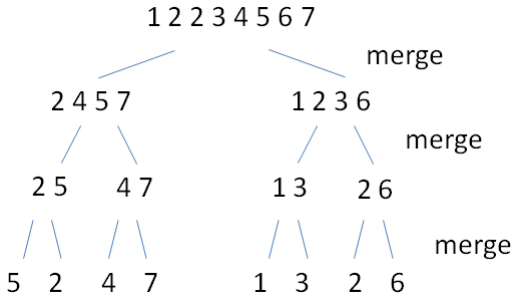


Figura 2.3: Ejecución algoritmo merge-Sort.

2.6.2 Analizando Algoritmos Recursivos

Recursión $\Rightarrow$ ecuación de recurrencia o "recurrencia".
$\Rightarrow$ tiempo de ejecución sobre un problema de tamaño n en términos de ejecución sobre la entrada más pequeña.

La recurrencia para un algoritmo divide y vencerás está basado en tres pasos.
Sea $T(n)$ el tiempo de ejecución sobre una entrada de tamaño n.
Si la entrada es muy pequeña $n \leq c$ para alguna c, la solución sencilla toma tiempo constante $\Theta(1)$.
Si la división toma a subproblemas de tamaño n/b (Para merge-sort $a = b = 2$)
$D(n)$: Tiempo de dividir
$C(n)$: Tiempo de combinar

$$T(n) = \begin{cases} \Theta(1) & \text{si } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{en otro caso.} \end{cases}$$

Para simplificar tomaremos n es 2^k
Consideremos el algortimo MergeSort en el peor caso Merge con 1 elemento $\Rightarrow \Theta(1)$
Cuando $n > 1$:
Divide $\rightarrow D(n) = \Theta(1)$
Conquer $\rightarrow D(n) = 2T(\frac{n}{2})$ dos problemas de la mitad del tamaño
Combinar $\rightarrow MergeO(n)$

$$T(n) = \begin{cases} c & \text{si } n = 1 \\ 2T(\frac{n}{2}) + cn & \text{si } n > 1. \end{cases}$$

Resolviendo la recurrencia:

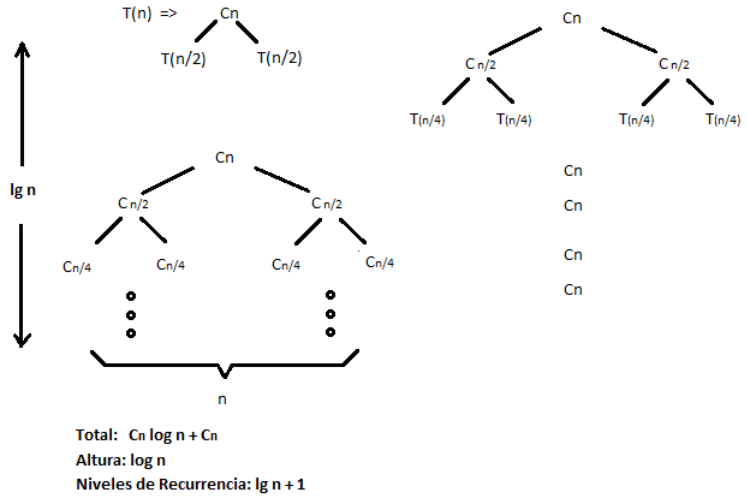


Figura 2.4: Tratamiento a la Recurrencia.

2.6.3 Resolviendo Recurrencias

En general, evitamos ciertos detalles técnicos cuando planteamos y resolvemos problemas (recurrencias).

Ejemplo 2.6.2 (Resolviendo la Recurrencia del Algoritmo MergeSort)**.**

$$T(n) = \begin{cases} \Theta(1) & si\ n = 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n) & si\ n > 1. \end{cases}$$

Ejemplo 2.6.3 (Condiciones de frontera)**.** $\underbrace{T(1) = T(0) + T(1) + \Theta(1)}$

sigue siendo constante

$\downarrow$

$\Theta(1) = \Theta(1)$

La secuencia puede cambiar pero en un valor constante.

Al resolver recurrencias se omiten pisos y celines (techos), y condiciones de frontera.

Resolviendo Recurrencias: Método de Sustitución

Procedimiento. Adivinar la forma de la solución, luego usar inducción matemática para mostrar que la solución es correcta.

$T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + n$

Suponemos $T(n) = O(n \lg n) \Rightarrow \boxed{T(n) \leq cn \lg n}$, constituyendo la HI (Hipótesis de Inducción)

TI (Tesis de Inducción)

$$\begin{aligned} T(n) &\leq 2(c\lfloor \tfrac{n}{2} \rfloor \log \lfloor \tfrac{n}{2} \rfloor) + n \\ &\leq cn(\lg n - 1) + n \\ &\leq cn \lg n - cn + n \\ &\leq cn \lg n \end{aligned}$$

Así empleando *HI* y *TI*, tenemos que:

$T(\lfloor \frac{n}{2} \rfloor) \leq c\lfloor \frac{n}{2} \rfloor \lg(\lfloor \frac{n}{2} \rfloor)$, para $\boxed{c \geq 1}$
Dado que la inducción matemática está incompleta, requiere mostrar que nuestra solución funciona para condiciones de frontera (casos base).
Debemos demostrar que c funciona también para las condiciones de frontera:
$T(1) = 1$ (Es la única condición de frontera)
Entonces para $n = 1 \qquad T(1) = c1\lg 1$
T(1)=0 (?)
Así el caso base falló!!! (pero este no es el fin)
Recordar que $T(n) = cn\lg n$ para $n \geq n_0$, donde n_0 es una constante.
Para $n > 3$ la recurrencia no depende de 1.
Reemplazaremos $T(2)$ y $T(3)$ como los casos base en lugar de $T(1)$ haciendo $n_0 = 2$
$T(1)$ caso base de la recurrencia.
$T(2)$ y $T(3)$ casos base de la prueba
$T(2) = c$
$T(2) \leq c2\lg 2$
$T(3) \leq c3\lg 3$

$$\left.\begin{array}{l} T(2) = 2T(1) + 2 = 4 \\ T(3) = 2T(1) + 3 = 5 \end{array}\right\} \begin{array}{l} 4 \leq c2\lg 2 \\ 5 \leq c3\lg 3 \\ c \geq 2 \text{ suficiente} \end{array}$$

Haciendo buenas "adivinaciones".
Experiencia + Creatividad.

Heurística 1. Árbol de recursión.
Ejemplo: $T(n) = 2T(\lfloor \frac{n}{2} \rfloor + 17) + n$
Heurística 2. Aproximación.

$$\begin{array}{c} \downarrow \Omega(n) \\ \ldots n\log n \ldots \Theta(n) \\ \uparrow O(n^2) \end{array}$$

A veces se necesita una hipótesis más fuerte:
$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 1 \Rightarrow T(n) \leq cn(sup)$
$T(n) = c(\lfloor \frac{n}{2} \rfloor) + c(\lceil \frac{n}{2} \rceil) + 1$
$T(n) = cn + 1$ (No se puede demostrar)
Hipótesis más fuerte:
$T(n) \leq cn - b$; $b \geq 0$ (No se puede demostrar)
$T(n) \leq (c\lfloor \frac{n}{2} \rfloor - b) + c(\lceil \frac{n}{2} \rceil - b) + 1$
$T(n) = cn - 2b + 1$
$T(n) \leq cn$

Resolviendo Recurrencias: Método: Árbol de Recursión

Éste método es útil para algoritmos del tipo "Divide y Conquistar".
El árbol de recursión está compuesto por:

- **Nodo**: Costo de un solo subproblema en alguna parte de la invocación recursiva
- **Costo por Nivel**: Suma de todos los costos en cada nodo del nivel
- **Costo Total**: Suma de los costos de todo el árbol

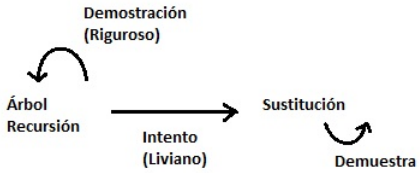


Figura 2.5: Método del Árbol de Recursión.

Ejemplo 2.6.4. *Resolver la recurrencia* $T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + \Theta(n^2)$ *(Usando Árbol de Recursión)*

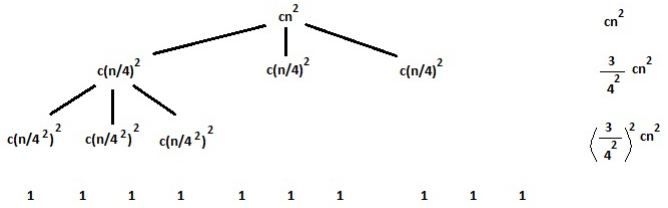


Figura 2.6: Árbol de Recursión para el Ejemplo Citado.

Costo de cada nivel del árbol

A la profundidad i el número de nodos es $3i$ *ya que cada subproblema divide a* 4 *a cada nodo en cada nivel* $i = 0,1,2,3,\ldots,\log_4 n - 1$*, tiene un costo de* $c(\frac{n}{4^i})^2$

$$3^i \cdot c(\tfrac{n}{4^i})^2 = (\tfrac{3}{16})^i cn^2$$

En el último nivel cada nodo vale 1

$$\begin{gathered} (\lg_4 n) \\ (1) \cdot 3^{\lg_4 n} = n^{\lg_4 3} \Rightarrow \Theta(n^{\log_4 3}) \\ 3^{\lg_4 n} = y \\ \lg_n 3 \lg_4 n = \lg_n y \end{gathered}$$

El Costo Total

$$\begin{aligned} T(n) &= cn^2 + \tfrac{3}{16}cn^2 + (\tfrac{3}{16})^2 cn^2 + \cdots + (\tfrac{3}{16})^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\ &= \sum_{i=0}^{\lg_4 n - 1} (\tfrac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \tfrac{(\frac{3}{16})^{\log_4 n} - 1}{(\frac{3}{16} - 1)} cn^2 + \Theta(n^{\lg_4 3}) \\ &\leq \sum_{i=0}^{\infty} (\tfrac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \\ &\leq \tfrac{1}{(1 - (\frac{3}{16}))} cn^2 + \Theta(n^{\log_4 3}) \\ &\leq \tfrac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\ &= O(n^2) \end{aligned}$$

La primera llamada recursiva gasta cn^2

$\therefore \Omega(n^2)$
$\therefore \Theta(n^2)$
Ahora apliquemos el método de sustitución:
$T(n) \leq 3T(\lfloor \frac{n}{4} \rfloor) + cn^2$
$\leq 3d(\lfloor \frac{n}{4} \rfloor)^2 + cn^2$
$\leq \frac{3dn^2}{16} + cn^2$
$\leq dn^2$

Resolviendo Recurrencias: Método del Teorema del Maestro

Considerando la recurrencia $T(n) = aT(\frac{n}{b}) + f(n)$; $a \geq 1$ y $b > 1$ donde $f(n)$ es una función asintóticamente positiva. El problema de tamaño n es dividido en a subproblemas de tamaño $\frac{n}{b}$ donde a y b son constantes positivas. Los a problemas son resueltos recursivamente en tiempo $T\frac{n}{b}$. El costo de dividir el problema y combinar los resultados en subproblemas es $f(n) = D(n) + C(n)$. Por ejemplo para el caso del algoritmo MergeSort se tiene que:

$$a = 2 \qquad b = 2 \qquad f(n) = \Theta(n)$$

Teorema 2.6.1 (Maestro)**.** *Sean $a \geq 1$ y $b > 1$ constantes, sea $f(n)$ una función y sea $T(n)$ definida para enteros no negativos como la recurrencia*

$$T(n) = aT(\tfrac{n}{b}) + f(n)$$

Donde interpretamos $\frac{n}{b}$ como $\lfloor \frac{n}{b} \rfloor$ o $\lceil \frac{n}{b} \rceil$. Luego $T(n)$ está acotada asintóticamente como sigue:

1. *Si $f(n) = O(n^{\log_b a - \varepsilon})$ para alguna constante $\varepsilon > 0$, entonces $T(n) = \Theta(n^{\lg_b a})$*
2. *Si $f(n) = \Theta(n^{\log_b a})$, entonces $T(n) = \Theta(n^{\log_b a} \lg n)$*
3. *Si $f(n) = \Omega(n^{\log_b + \varepsilon})$ para alguna constante $\varepsilon > 0$ y si $af(\frac{n}{b}) \leq cf(n)$, para alguna constante $c < 1$ y para todo n lo suficientemente grande, entonces $T(n) = \Theta(f(n))$*

Ejemplo 2.6.5. *Considere $T(n) = 9T(\frac{n}{3}) + n$*
Por tanto $a = 9$, $b = 3$ y $f(n) = \Theta(n)$
Evaluemos $f(n) = \Theta(n)$ comparado respecto a $O(n^{\log_b a \pm \varepsilon})$
$= \Theta(n)$ *comparado respecto a* $O(n^{2 \pm \varepsilon})$
Para $= \Theta(n) = O(n^{2-1})$; $\varepsilon = 1 > 0$
Por tanto $T(n) = O(n^2)$

Ejemplo 2.6.6. $T(n) = T(\frac{2^n}{3}) + 1$
Por tanto $a = 1$, $b = 3$ y $f(n) = 1$
1 *comparado respecto a* $O(n^{\lg_3 1 \pm \varepsilon})$; $\varepsilon = 0$
1 *comparado respecto a* $O(1)$
1 *comparado respecto a* $\Theta(1)$
$T(n) = n^{\log_b a} \lg n$
$T(n) = 1 \lg n$
$T(n) = \Theta(\lg n)$

Ejemplo 2.6.7. $T(n) = 3T(\frac{n}{4}) + n \lg n$
Por tanto $a = 3$, $b = 4$ y $f(n) = n \lg n$
$n \log n$ *comparado respecto a* $O(n^{0,79-\varepsilon})$ $\quad X$
$\lg_4 3 = 0{,}79$
$n \log n$ *comparado respecto a* $\Omega(n^{0,79+\varepsilon})$ $\quad \varepsilon > 0$
Para un n suficientemente grande:

$af(\frac{n}{b}) \leq cf(n)$
$a\frac{n}{b}\log\frac{n}{b} \leq cn\lg n$
$\frac{3}{4}n\log\frac{n}{4} \leq cn\lg n$
para $c \geq \frac{3}{4}$ *esto es cierto.*
$\therefore T(n) = \Theta(n\log n)$

3 — Funciones Recursivas

3.1 Funciones Recursivas Primitivas

3.1.1 Introducción

Intentaremos construir modelos que nos acerquen a comprender los fundamentos del cálculo y su posibilidad de modelizarlo.

No es fácil encontrar una definición del concepto “cálculo” que aparece en múltiples aspectos de nuestra vida.

Calcula quien efectua una multiplicación, pero también decimos que realiza un cálculo un jugador cuando patea una pelota.

Cada cálculo implica el realizar, operando sobre ciertos datos, una serie de pasos, para obtener como resultado una determinada información. Esto es, la aplicación de un **algorítmo.**

Trataremos de encontrar elementos básicos en los que se pueda “descomponer” cualquier cálculo.

Así como la química muestra que las múltiples sustancias que nos rodean son en su esencia combinación de un conjunto relativamente pequeño de átomos (y además la física nos dice que cada uno de esos átomos está formado por partículas elementales), se trata de ver si sucede algo parecido con respecto a lo que denominamos cálculo.

Estas ideas y los modos de desarrollarlas, elaboradas por lógicos y matemáticos, hasta mediados de los años cuarenta del siglo pasado, sólo interesaban a un limitado grupo de científicos que se dedicaban a esos temas.

Pero, con la creación de la computadora (que, justamente, es un instrumento que puede realizar millones de operaciones sencillas en segundos) han cobrado notable importancia.

Una buena aproximación es tratar de modelizar, con elementos básicos, el cálculo matemático.

En ese sentido, la teoría indica que basta encontrar un modelo que abarque el cálculo sobre los números naturales.

Lo que presentaremos a continuación es un modelo desarrollado principalmente a partir de trabajos de Dedekind, Skolem, Peano, Hilbert, Ackerman y Church denominado “Funciones Recursivas". (fue en 1931 cuando Gödel propone formalmente el concepto de recursividad para funciones).

3.1.2 Funciones Numéricas

Definición 3.1.1. *Llamaremos función numérica a toda función f cuyo dominio sea una potencia cartesiana de $\mathbb{N}_0$ y cuyo codominio sea $\mathbb{N}_0$,*

$$f : \mathbb{N}_0^k \to \mathbb{N}_0 \quad k \in \mathbb{N}_0.$$

Consideramos el caso en que el valor de k sea cero, con la convención de identificar una función de cero variables con un número perteneciente a los $\mathbb{N}_0$.

Una k-upla de elementos que pertenecen a $\mathbb{N}_0$ la notaremos $(x_1, x_2, ..., x_k)$.

También presentaremos las k-uplas de $\mathbb{N}_0^k$, para cada $k \in \mathbb{N}_0$, con letras mayúsculas como X, Y, Z. Cuando se quiera remarcar el hecho de que tienen k componentes, escribiremos X^k.

El valor que toma una función f en $(x_1, x_2, ..., x_k)$ se indicará $f(x_1, x_2, ..., x_k)$ o directamente $f(X)$

Si el dominio de una función es $\mathbb{N}_0^m$ diremos que es de *orden m* .

Cuando queremos hacer explícita referencia al orden de una función agregaremos al notarla un superíndice , o sea $f^{(m)}$ indica que f es de orden m. .

3.1.3 Funciones base

Definición 3.1.2. *Llamaremos funciones ceros a las funciones*

$$c^{(k)} : \mathbb{N}_0^{(k)} \to \mathbb{N}_0$$

$$\forall X \in \mathbb{N}_0^{(k)} \quad \mapsto c^{(k)}(X) = 0$$

Son funciones muy simples, ya que sólo consisten en asociar a cualquier valor el número cero.

Definición 3.1.3. *Definiremos una familia de funciones.*

Dados $n \in \mathbb{N}$, $k \in \mathbb{N}$ tales que $1 \leq n$, y $1 \leq k \leq n$.

Llamamos función proyección de orden n, en la k-ésima componente y notaremos

$$p_k^{(n)} : \mathbb{N}_0^n \to \mathbb{N}_0$$

$$(x_1, x_2, ..., x_n) \mapsto p_k^{(n)}(x_1, x_2, ..., x_n) \stackrel{def}{=} x_k$$

Se trata de una familia de funciones sencillas de calcular ya que seleccionan en una n-upla la k-ésima componente

Definición 3.1.4. *Llamamos función sucesor a la función*

$$s : \mathbb{N}_0 \to \mathbb{N}_0$$

$$x \in \mathbb{N}_0 \mapsto s(x) \stackrel{def}{=} x + 1$$

Es decir, s produce el sucesor de su valor de entrada. También en este caso, se trata de una función elemental.

Definición 3.1.5. *Llamaremos **funciones base** a las funciones $c^{(k)}$ con $k \geq 0$, a las proyecciones $p_k^{(n)}$ con $n, k \in \mathbb{N}$ tal que $1 \leq k \leq n$ y a la función sucesor $s^{(1)}$.*

La idea es agregar herramientas de construcción que nos permitan combinando estas funciones bases, como ladrillos elementales, obtener funciones más complejas.

3.1.4 Operador Composición

El modo más inmediato de combinar funciones es "encadenándolas " de modo que los resultados de la aplicación de un grupo de ellas sean del dominio de otra.

Definición 3.1.6. *Dados $n \geq 1$, $k \geq 0$, una función numérica $f^{(n)}$ y una familia de n funciones numéricas de orden k , $\{g_i^{(k)}\}_{i=1}^n$, diremos que la función h tal que*

$$h : \mathbb{N}_0^k \to \mathbb{N}_0$$

$$X \in \mathbb{N}_0^k \mapsto \ \ h(X) \stackrel{def}{=} f(g_1(X), g_2(X), ..., g_n(X)) \in \mathbb{N}_0$$

es definida por la composición de las funciones $f^{(n)}$, $\{g_i^{(k)}\}_{i=1}^n$ y notaremos:

$$h = \Phi\left(f^{(n)}, g_1^{(k)}, g_2^{(k)}, ..., g_n^{(k)}\right)$$

En algunos casos al operador se le agregan los índices $n+1$ y k, $\quad \Phi_k^{n+1}$, para hacer referencia a las $n+1$ funciones sobre las que opera, y a las k-uplas de su dominio.

Φ_k^{n+1} no está definido para todas las $n+1$-uplas de funciones, sino sólo para aquellas n+1-uplas en las cuales la primera sea una función de de n-variables y las n restantes sean de orden k.

Ejemplo 3.1.1. *Podemos construir una familia de funciones constantes $uno^{(k)}$ que asocia a toda k-upla el valor* 1

$$uno^{(k)} = \Phi(s, c^{(k)})$$

Así también podemos construir cualquier función constante:
Por ejemplo $tres^{(k)}$ que asocia a cada k-upla el numero tres

$$tres^{(k)} = \Phi\left(s, \Phi\left(s, \Phi\left(s, c^{(k)}\right)\right)\right)$$

Ejercicio para pacientes: construir la constante $cuarentidos^{(k)}$ ....

Ejemplo 3.1.2. *Componiendo la función sucesor, podemos contruir funciones sobre $\mathbb{N}_0$ que devuelvan un número sumado a cualquier valor.*
Así por ejemplo la función $Mas5^{(1)}$

$$Mas5^{(1)} = \Phi(s, (\Phi(s, \Phi(s, (\Phi(s, s))))))$$

Ejercicio para muy pacientes: construir la función $Mas258$ ....

Sin embargo utilizando como elementos de construcción sólo las funciones base y el operador Φ no podemos conseguir funciones de una cierta complejidad.

Se puede ver, por ejemplo que no es posible, usando unicamente estos recursos construir una función sobre $\mathbb{N}_0$ llamada *doble* tal que duplica un valor.

$$doble(x) = x + x. \text{ (Se recomiendo intentarlo)}$$

Se hace necesario introducir nuevas herramientas de construcción.

3.1.5 Operador Recursión

Definición 3.1.7. *Definiremos un nuevo operador R que llamaremos* recursión *que, a partir de dos funciones:* $g^{(k)}$ *y* $h^{(k+2)}$ *, construye una nueva función* $f^{(k+1)}$

$$f^{(k+1)} = R(g^{(k)}, h^{(k+2)})$$

definida para cada k+1-upla, del siguiente modo:
Si el primer elemento de la $k+1$*upla es igual a cero:*

$$f(0, X^k) \stackrel{def}{=} g(X^k)$$

Si el primer elemento es distinto de cero (lo presentamos como $y+1$*) el valor de la función se calcula recursivamente del modo siguiente:*

$$f(y+1, X^k) \stackrel{def}{=} h(y, X^k, f(y, X^k))$$

Estamos ahora en condiciones de definir al conjunto de funciones recursivas primitivas en forma inductiva :

Definición 3.1.8. *Definiremos el conjunto de Funciones Recursivas Primitivas de la siguiente manera:*

a) Las funciones base definidas en 3.1.5 son Funciones Recursivas Primitivas.
b) Las funciones obtenidas a partir de Funciones Recursivas Primitivas aplicándo un número finito de operaciones de composición y recursón(Φ *y* R *) son Funciones Recursivas Primitivas.*

Notaremos a este conjunto : ***FRP****.*

Ejemplo 3.1.3. *Son* ***FRP*** *las funciones* $\Sigma^{(2)}$, $\Pi^{(2)}$, $Fac^{(1)}$, $Exp^{(2)}$, $Pd^{(1)}$ *definidas del siguiente modo:*

I) $\Sigma(y,x) \stackrel{def}{=} y+x$ $(Suma)$
II) $\Pi(y,x) \stackrel{def}{=} y.x$ $(Producto)$
III) $Fac(x) \stackrel{def}{=} x!$ $(Factorial)$
IV) $Exp(y,x) \stackrel{def}{=} x^y$ *(Exponencial) con la convención de que* $0^0 = 1$
V)

$$Pd(x) \stackrel{def}{=} \begin{cases} 0 & \text{si } x = 0 \\ x-1 & \text{si x} > 0 \end{cases} \quad (Predecesor)$$

Veamos cómo se muestra i), que $\Sigma(x,y) \in$ ***FRP***
En el caso de que la primer componente sea 0 *es:*

$$\Sigma(0,x) = 0+x = x = p_1^{(1)}(x) \quad luego \quad la \quad g^{(1)} = p_1^{(1)}$$

Si la primer componente es mayor que cero:

$$\Sigma(y+1,x) = (y+1)+x = (y+x)+1 = s(\Sigma(y,x)) = s(p_3^{(3)}(y,x,\Sigma(y,x)))$$

$$entonces \quad h^{(3)} = \Phi(s, p_3^{(3)})$$

De esto deducimos que se puede construir Σ *por recursión a partir de* $p_1^{(1)}$ *y* $\Phi(s, p_3^{(3)})$*; en efecto, tenemos que*

$$\Sigma = R(p_1^{(1)}, \Phi(s, p_3^{(3)}))$$

Veamos la función definida en v), Pd (Predecesor)

$$Pd(0) = 0 = c^{(0)}$$

y además

$$Pd(y+1) = y = p_1^{(2)}(y, f(y))$$

De esto deducimos que se puede construir Pd por recursión a partir de $c^{(0)}$ *y* $p_1^{((2)}$

$$Pd = R(c^{(0)}, p_1^{(2)})$$

Se dejan las demostraciones para las funciones ii), iii), iv) como ejercicio.

Observación importante: En esta parte del libro, aplicamos el criterio extensional considerando que dos funciones, f, g son iguales si tienen el mismo dominio y para todo elemento X que pertenezca a dicho dominio, $f(X) = g(X)$, aunque utilicemos distintos procedimientos para calcularlas. No nos preocupa la eficiencia del cálculo. Si una función pertenece a **FRP** todas las combinaciones de funciones base que la construyan son equivalentes.

Mostraremos a continuación una serie de propiedades que seran útiles para encontrar nuevos miembros de la familia **FRP**.

Comencemos con la llamada potencia n de una función f de orden 1, que es la función que obtenemos aplicando n veces la función f, con la convención que aplicada 0 veces es la identidad.

Esto nos permite pensar la potencia de una función como una función de dos variables.

Definición 3.1.9. *Dada una función* $f^{(1)}$ *definimos una nueva función* $F^{(2)}$*, llamada* ***potencia*** *de f, del siguiente modo:*

$$F(0,x) = x \quad \forall x \in \mathbb{N}_0$$

$$F(y+1,x) = f(F(y,x)) \quad \forall x \in \mathbb{N}_0$$

Notaremos $F(y,x)$ *como* $f^y(x)$

Proposición 3.1.1. *Dada* $f^{(1)}$*, sea* $F(y,x) = f^y(x)$ *(potencia de f) , entonces si*

$$f \in \textbf{\textit{FRP}} \Longrightarrow F \in \textbf{\textit{FRP}}$$

Demostración.

$$F(0,x) = f^0(x) = x = p_1^{(1)}(x)$$

$$F(y+1,x) = f(f^y(x)) = f(p_1^{(3)}((y,x,F(y,x))))$$

entonces

$$F = R\left[p_1^{(1)}, \Phi(f, p_3^{(3)})\right]$$

■

Ejemplo 3.1.4. *Definiremos una función $\hat{d}^{(2)}$ del siguiente modo:*

$$\hat{d}^{(2)}(y,x) \stackrel{def}{=} \begin{cases} x-y & \text{si } x \geq y \\ 0 & \text{si x} < \text{y} \end{cases}$$

Se trata de una resta, que se realiza cuando es posible. En algunos casos notaremos a $\hat{d}^{(2)}(y,x)$ como $x \dot{-} y$

*Se ve que la función $\hat{d}^{(2)}$ pertenece a las **FRP** ya que se puede presentar como la potencia de la función Pd*

$$\hat{d}(y,x) = Pd^{y}(x)$$

y aplicamos la proposición 3.1.1.

*La misma proposición se puede utilizar para mostrar que $\Sigma^{(2)}$ pertenece a las **FRP** ya que se puede ver que es la potencia del sucesor:*

$$\Sigma(y,x) = s^{y}(x)$$

Ejemplo 3.1.5. *Definiremos una función $D_{\mathbf{0}}^{(1)}$ del siguiemte modo:*

$$D_{\mathbf{0}}^{(1)}(y) \stackrel{def}{=} \begin{cases} 1 & \text{si } y = 0 \\ 0 & \text{si y} > 0 \end{cases}$$

Se trata de una función que distingue al cero de los demás números.

*Se puede ver que $D_{\mathbf{0}}^{(1)} \in$ **FRP** ya que:*

$$D_{\mathbf{0}}^{(1)} = R[\Phi(s, c^{(0)}), c^{(2)}]$$

Teorema 3.1.1. *Sea $f^{(2)}$ Sean*

a) $F(y,x) = \sum_{z=0}^{y} f(z,x)$

b) $G(y,x) = \prod_{z=0}^{y} f(z,x)$

*Entonces si $f \in$ **FRP** $\Longrightarrow F, G \in$ **FRP***

Demostración de a).

$$F(0,x) = f(0,x) = f(c^{1}(x),x)$$

$$F(y+1,x) = \Sigma(F(y,z), F(y+1,x)) =$$

$$\Sigma(p_1^{(3)}(F(y,x),y,x), f(s(p_2^{(3)}(F(y,x),y,x), p_3^{(3)}(F(y,x),y,x)))$$

Por lo tanto:

$$F^{(2)} = R\left[\phi[f^{(2)}, c^{(1)}, p_1^{(1)}], \phi[\Sigma, p_1^{(3)}, \phi[f^{(2)}, \phi[s^{(1)}, p_2^{(3)}], p_3^{(3)}]]\right]$$

*O sea F se construye aplicando los operadores ϕ y R a las funciones: f (por hipótesis $\in$ **FRP**), $\Sigma \in$ **FRP** segun 3.1.3 y funciones base (que son **FRP** por definición) , luego $F \in$ **FRP***

Se trata de una especie de integral discreta.

Para la parte b) se puede proceder analogamente sustituyendo Σ por Π

$$G^{(2)} = R\left[\phi[f^{(2)}, c^{(1)}, p_1^{(1)}], \phi[\Pi, p_1^{(3)}, \phi[f^{(2)}, \phi[s^{(1)}, p_2^{(3)}], p_3^{(3)}]]\right]$$

En forma similar se puede mostrar que dada una $f^{(n+1)} \in$ ***FRP*** *las funciones* $F^{(n+2)}$ *y* $G^{(n+2)}$ *definidas como:*

$$F(y,X) = \sum_{(z=0)}^{y} f(z,X)$$

$$G(y,X) = \prod_{(z=0)}^{y} f(z,X)$$

tambien son ***FRP***

3.1.6 Conjuntos recursivos primitivos (CRP)

Definición 3.1.10. *Dado un conjunto X, para cada subconjunto* $A \subseteq X$ *definimos su función característica*

$\chi_A : X \to \{0,1\}$ *del siguiente modo:*

$$\chi_A(x) = \begin{cases} 1 & \text{si } x \in A \\ 0 & \text{si } x \notin A \end{cases}$$

La función característica de un conjunto lo determina totalmente. Recíprocamente, dada una función $\chi : X \to \{0,1\}$, existe un único subconjunto A de X tal que $\chi_A = \chi$.

Esto nos permite establecer una clasificación en los subconjuntos de $\mathbb{N}_0^k$

Definición 3.1.11. *Diremos que un subconjunto A de* $\mathbb{N}_0^k$ *es un* conjunto recursivo primitivo, *notaremos* ***CRP*** *si su función característica* $\chi_A : \mathbb{N}_0^k \to \{0,1\}$ *es recursiva primitiva.*

Proposición 3.1.2. *Sea* $k \in \mathbb{N}$, *y sean A y B dos subconjuntos de* $\mathbb{N}_0^k$. *Si A ,B son* ***CRP*** *, el complemento* $\neg A$ *, la intersección (* $A \cap B$*) y la unión (*$A \cup B$*) son* ***CRP*** *.*

Demostración. Como A, B son **CRP** se tiene que sus funciones características χ_A , $\chi_B \in$ **FRP**

(a) $\chi_{\neg A} = \Phi(D^0, \chi_A)$

(b) $\chi_{A \cap B} = \chi_A \cdot \chi_B = \Phi(\Pi, \chi_A, \chi_B)$

En a) y b) las funciones características se construyen a partir de **FRP**, por lo tanto sus conjuntos asociados son **CRP**.

(c) $\chi_{A \cup B}$

Recordemos que $A \cup B = \neg[(\neg A) \cap (\neg B)]$ (De Morgan)

Aplicamos, entonces, lo visto en a) y b).

■

3.1.7 Relaciones recursivas primitivas (RRP)

En el discurso matemático las funciones constituyen un caso particular de las llamadas relaciones entre conjuntos.

Ante una expresión como "$a + b \leq c$" se puede, para cada terna $(x_1, x_2.x_3)$ de $\mathbb{N}_0^3$ verificar si sus elementos cumplen o no con la expresón.

Se trata en este caso de una relación sobre $\mathbb{N}_0^3$.

En el mismo sentido una afirmación como "x es primo " es una relación definida sobre $\mathbb{N}_0^1$

Cada relación sobre $\mathbb{N}_0^k$ queda completamente determinada por el conjunto de las k-uplas que verifican la relación. (decimos que la relación es **verdadera** en esas k-uplas)

Es entonces que dada una relación α definida sobre $\mathbb{N}_0^k$ podemos asociar a la misma el subconjunto de $\mathbb{N}_0^k$, de los valores en que la relación es verdadera. Notaremos $D_\alpha \subset \mathbb{N}_0^k$ a dicho conjunto.

Así por ejemplo dada la relación $x = y$ el subconjunto asociado $D_=$ es la diagonal de $\mathbb{N}_0^2$

A partir de la asociación de una *relación* con el *conjunto de los valores donde es verdadera* definimos:

Definición 3.1.12. *Dada una relación α definida en $\mathbb{N}_0^k$ diremos que es Relación Recursiva Primitiva (notaremos **RRP**) si su conjunto asociado D_α es **CRP***

Ejemplo 3.1.6.

Mostraremos que la relación '=' es recursiva primitiva.

Llamaremos $D_=$ al conjunto asociado a esta relación.

$$D_= = \{(x,x)\ \ x \in \mathbb{N}_0\}$$

.

Observemos que la función $E(x,y) = D_{\mathbf{0}}^{(1)}(x \dot{-} y + y \dot{-} x)$ es tal que sólo vale 1 si $x = y$ y vale 0 para cualquier otro caso.

Esta es la función característica del conjunto $D_=$.

*Es inmediato que esta función es **FRP**, por lo que el conjunto $D_=$ es **CRP** y resulta la relación '=' es **RRP**.*

Definición 3.1.13. *Sean α, β relaciones definidas sobre $\mathbb{N}_0^k$*

Definimos :

I) *$\neg\alpha(X)$ la relación que es verdadera $\Leftrightarrow$ $\alpha(X)$ es falsa*

II) *$(\alpha \wedge \beta)(X)$ la relación que es verdadera $\Leftrightarrow$ $\alpha(X)$ y $\beta(X)$ son ambas verdaderas*

III) *$(\alpha \vee \beta)(X)$ la relación que es verdadera $\Leftrightarrow$ $\alpha(X)$ o $\beta(X)$ son verdaderas*

Proposición 3.1.3.

*Si α, β son **RRP**, $\Longrightarrow$ $\neg\alpha,\ \alpha \wedge \beta,\ \alpha \vee \beta\ \in$ **RRP***

La demostración es inmediata a partir de la proposición 3.1.2 .

Definición 3.1.14. *Dada una relación α definida en $\mathbb{N}_0^{k+1}$*

I) *Llamaremos cuantificador universal y lo notaremos $\bigwedge_y \alpha(y,X)$ a la relación que depende sólo de X que es verdadera si es verdadera la relación $\alpha(y,X)$ para todo y.*

II) *Llamaremos cuantificador universal limitado y lo notaremos: $\bigwedge_{y=0}^{z} \alpha(y,X)$ a la relación, que depende de X y de z, que es verdadera si es verdadera la relación $\alpha(y,X)$ para todos los valores de y no superiores a z.*

III) *Llamaremos cuantificador existencial, y lo notaremos $\bigvee_y \alpha(y,X)$ a la relación que depende sólo de X que es verdadera si es verdadera la relación $\alpha(y,X)$ para al menos un valor de y.*

IV) *Llamaremos cuantificador existencial limitado y lo notaremos: $\bigvee_{y=0}^{z} \alpha(y,X)$ a la relación, que depende de X y de z, que es verdadera si es verdadera la relación $\alpha(y,X)$ para al menos un valor y no superior a z.*

Proposición 3.1.4. *Sea la relación α definida en $\mathbb{N}_0^{k+1}$*

*Si α es **RRP** $\Longrightarrow$ $\gamma(z,X) = \bigwedge_{y=0}^{z} \alpha(y,X)$, $\delta(z,X) = \bigvee_{y=0}^{z} \alpha(y,X)$ son **RRP**.*

Demostración:

Sea f_{D_α} la función característica asociada a la realación α

Entonces la función asociada a γ llamada f_{D_γ} será :

$$f_{D_\gamma(z,X)} = \prod_{z=0}^{y} f_{D_\alpha(z,X)}$$

*Por la parte b) del teorema 3.1.1 vimos que esta función es **FRP** dado que f_{D_α} es **FRP**. Para la demostración de la parte ii) debemos, nuevamente, apelar a De Morgan . Observemos que*

$$D_\delta(z,X) \equiv \neg \bigcup_{y=0}^{z} \neg\, D_\alpha(y,X)$$

y la demostración sale inmediatamente aplicando los resultados anteriores.

Atención: Estos resultados no se pueden extender, en general, para el caso de los cuantificadores no limitados.

Veamos algunos ejemplos de **RRP**.

Ejemplo 3.1.7.
$x \neq y \in$ ***RRP***
inmediato pues $(x \neq y) \equiv \neg(x = y)$

Ejemplo 3.1.8.
$x \geq y \in$ ***RRP*** *en efecto:*

$$x \geq y \equiv \bigcup_{z=0}^{x} (y = z)$$

Ejemplo 3.1.9.
$x > y \in$ ***RRP*** *pues*

$$x > y \equiv (x \geq y) \wedge \sim (x = y)$$

Ejemplo 3.1.10.
La relación $Mult(x,y)$ *(x es múltiplo de y) es **RRP***
(la relacion es verdadera si $\exists k \in \mathbb{N}_0$ *tal que* $x = k.y$ *)*
En efecto:

$$Mult(x,y) \equiv \bigcup_{z=0}^{x} (x = z.y)$$

Ejemplo 3.1.11.
Sea relación "x es un número primo" .(Notaremos $Prim(x)$*)*
Entonces $Prim(x) \quad \in$ ***RRP***
Sabemos que un valor x es primo si es $x > 1$ *y no es múltiplo de ningún nḿero z tal que* $1 < z < x$ *.*

$$Prim(x) \equiv (x > 1) \wedge \bigcap_{z=2}^{x-1} (\neg Mult(x,z)))$$

Enunciaremos ahora una proposición, que será de aplicación para las funciones definidas por casos en una partición.

Proposición 3.1.5.

Sean los conjuntos recursivos primitivos $A_1, A_2, ...A_p \subset \mathbb{N}_0^m$ *una partición de* $\mathbb{N}_0^m$

$$\bigcup_{i=1}^{i=p} A_i = \mathbb{N}_0^m \quad y \quad \forall i \neq j \quad A_i \cap A_j = \emptyset$$

Entonces si $g_1^{(m)}, g_2^{(m)}, ...g_p^{(m)}$ *son funciones recursivas primitivas, la función* $h^{(m)}$ *definida por*

$$h(X) \stackrel{def}{=} \begin{cases} g_1(X) & \text{si } X \in A_1 \\ g_2(X) & \text{si } X \in A_2 \\ & ..... \\ g_p(X) & \text{si } X \in A_p \end{cases}$$

es recursiva primitiva.

Para demostrarlo basta observar que como los conjuntos A_i *son* ***CRP*** *implica que sus funciones características* χ_{A_i} *son* ***FRP****.*

De acuerdo a la definición f *se puede escribir como:*

$$f(X) = \sum_{i=1}^{n} g_i(X).\chi_{A_i} \quad \Rightarrow f \in \textbf{\textit{FRP}}$$

Definición 3.1.15. *Sea una relación* α *definida en* $\mathbb{N}_0^{k+1}$

Usaremos la notación $\mu(\alpha(y, X^k))$ *para indicar el mínimo valor de y para el que es verdadera la relación* $\alpha(y, X^k)$

Tal símbolo sólo tiene sentido si la relación se verifica para algún valor de y. Decimos que se cumple la condición de regularidad cuando eso sucede.

Teorema 3.1.2. *Sea una relación* α *definida en* $\mathbb{N}_0^{k+1}$ *tal que se cumple la condición de regularidad para todo elemento de* $\mathbb{N}_0^{k+1}$

O sea que para cada X^k *existe un valor de y tal que* $\alpha(y, X^k)$ *es verdadera*

Podemos definir

$$f(X) = \mu(\alpha(y, X^k))$$

Definimos la relación β *sobre* $\mathbb{N}_0^{k+1}$ *del siguiente modo:*

$$\beta(y, X) \equiv (y = f(X))$$

Entonces si $\alpha \in$ ***RRP*** $\Longrightarrow \beta \in$ ***RRP***

En efecto se tiene:

$$\beta(y, X) \equiv \bigcap_{z=0}^{y} (z < y \wedge (\neg(\alpha(z, X))) \vee (z = y) \wedge (\alpha(z, X))$$

Teorema 3.1.3.

Si además de verificarse las condiciones del teorema anterior existe una $g^{(k)} \in$ ***FRP*** *tal que para cada X,* $f(X) \leq g(X)$

Entonces podemos afirmar que la $f \in$ ***FRP***

En efecto, podemos escribir:

$$f(X) = \sum_{y=0}^{g(X)} y.D_\alpha(y,X)$$

Ejemplo 3.1.12.
Sea $Q(x,y) = [x/y]$, parte entera de la división.
(Con la convención $Q(x,0) = 0$.)
*Entonces $Q \in$ **FRP***
En efecto :

$$Q(x,y) = \mu z((y = 0) \vee ((z+1).y > x))$$

Ejemplo 3.1.13.
Sea $Mod(x,y)$ el resto de la división de x por y.
(Con la convención $Mod(x,0) = 0$)
*Entonces $Mod(x,y)$ es **FRP***
En efecto

$$Mod(x,y) = (x \dot{-} (y.Q(x,y))$$

Ejemplo 3.1.14. *Sea la función $N^{(1)}$ que a cada valor x le asocia el x-ésimo número primo.*
Así $N(0) = 2$, $N(1) = 3$, $N(2) = 5$.
*Mostraremos que esta singular función es **FRP**.*
Utilizaremos la función Prim definida en 3.1.11
Definimos la función h que dado un valor x devuelve el mínimo número primo mayor que x

$$h(x) = \mu\, y[Prim(y) \wedge (y > x)]$$

Por ejemplo h(8)=11.

*h está acotada por $Fac(x)+1$ que es **FRP**, puesto que se puede mostrar que siempre existe un número primo p tal que $x < p < x! + 1$*
*Entonces como está acotada por una **FRP** aplicando el teorema 3.1.3, h es **FRP**.*
Entonces definimos $N(x)$ como la potencia x de h aplicada a 2.

$$N(x) = h^x(2)$$

*N es **FRP***

3.1.8 Función Dupla

Definición 3.1.16. *Llamaremos **Función Dupla**, una función, $W^{(2)}$, que tenga las siguientes propiedades:*

I) $W(x,y) = W(m,n) \implies x = m \ , \ y = n$ *(unicidad)*
II) $x > m \implies W(x,y) > W(m,y)$ *(creciente respecto a la primera componente)*
III) $y > n \implies W(x,y) > W(x,n)$ *(creciente respecto a la segunda componente)*

A partir de estas propiedades se puede observar que si W es función dupla entonces

$$W(x,y) \geq x \quad W(x,y) \geq y$$

Por la propiedad i) vemos que podemos definir las inversas $U^{(1)}$ y $V^{(1)}$ tales que:

$$z = W(x,y) \implies U(z) = x \quad V(z) = y$$

y se cumple

1.
$$W(U(z),V(z)) = z$$

2.
$$U(W(x,y)) = x$$

3.
$$V(W(x,y)) = y$$

Observemos que por satisfacer las condiciones ii) e iii) de la definición 3.1.16 las funciones U y V pueden definirse:

$$U(z) = \mu x(\bigcup_{y=0}^{z} z = W(x,y)) \ , U(z) \leq z$$

$$V(z) = \mu y(\bigcup_{x=0}^{z} z = W(x,y)) \ , V(z) \leq z$$

y por lo tanto aplicando 3.1.3

$$\text{Si } W \in \mathbf{FRP} \Longrightarrow \quad U, V \in \mathbf{FRP}$$

Ejemplo 3.1.15. *Se puede verificar que son funciones **dupla** las siguientes:*

a)
$$W(x,y) = 2^x.(2y+1)\dot{-}1$$

b)
$$W(x,y) = \left[\frac{(x+y)(x+y+1)}{2}\right] + x$$

*Es inmediato que ambas son **FRP***

Veamos una aplicacion de estas funciones:

Teorema 3.1.4. *Sean $g_1^{(n)}$, $g_2^{(n)}$, $h_1^{(n+3)}$, $h_2^{(n+3)}$*

*Decimos que $f_1^{(n+1)}$ y $f_2^{(n+1)}$ se construyen por **Recursión entrelazada** si son definidas del siguiente modo:*

$$f_1(0,X) = g_1(X)$$

$$f_1(y+1,X) = h_1(f_1(y,X), f_2(y,X), y, X)$$

$$f_2(0,X) = g_2(X)$$

$$f_2(y+1,X) = h_2(f_2(y,X), f_1(y,X), y, X)$$

*Entonces si $g_1^{(n)}$, $g_2^{(n)}$, $h_1^{(n+3)}$, $h_2^{(n+3)}$ $\in$ **FRP** $\Longrightarrow f_1^{(n+1)}$ y $f_2^{(n+1)} \in$ **FRP***

Demostración. Sea W, una función **dupla** y **FRP** (como las del ejemplo 3.1.15)

Sea $F(y,X) = W(f_1(y,X), f_2(y,X))$

Mostraremos que $F \in$ **FRP**

$$F(0,X) = W(g_1(X), g_2(X)) \qquad g = \Phi(W, g_1, g_2)$$
$$F(y+1,X) = W(h_1(f_1(y,X), f_2(y,X), y, X), h_2(f_2(y,X), f_1(y,X), y, X)) =$$

$$F(y+1,X) = W(h_1(U(F(y,X)), V(F(y,X)), y, X), h_2(V(F(y,X)), U(F(y,X)), y, X)$$

Como F se puede definir por recursión de funciones **FRP** $\Longrightarrow F \in$ **FRP**

Pero por construcción:

$$f_1(y,X) = U(F(y,X)) \quad f_2(y,X) = V(F(y,X)$$

Luego f_1 y f_2 son **FRP**

■

3.2 Las FRP no alcanzan...

3.2.1 Introducción

En la sección anterior hemos presentado una gran cantidad de ejemplos de cálculos que se pueden representar con las **FRP**.

Sin embargo mostraremos que existen funciones que podemos calcular (esto es existe un procedimiento que nos permite encontrar la imagen de la función para determinados valores), que no pueden ser **FRP**.

Observemos, para empezar, que podemos definir funciones sobre los naturales que no son totales.

Así por ejemplo, la función natural "raíz cuadrada " sólo esta definida para algunos elementos de $\mathbb{N}_0$

Un camino que que podemos utilizar para calcularla. es por ejemplo:

$$\begin{array}{l} Leer\ \ (n) \\ R \leftarrow 0 \\ Mientras\ \ (R*R) \neq n \quad hacer\ \ R \leftarrow R+1 \\ Mostrar\ \ (R) \end{array}$$

Este procedimiento "funciona " sólo para algunos valores de n. Si la entrada es un número que no es un cuadrado perfecto, el procedimiento no termina. (se trata de un semialgoritmo).

Le podemos, entonces, asociar una función definida para una parte de los naturales (se trata de una función parcial).

Pero este cálculo no podrá ser representado por una **FRP** ya que, estas son siempre totales como mostraremos en el siguiente teorema.

Teorema 3.2.1. *Si f es **FRP** entonces f es total.*

Demostración. En esta demostración aprovecharemos la forma de construcción inductiva de las **FRP**

Se muestra que la propiedad se cumple para las funciones base (a) y luego se muestra que la propiedad se "hereda" al realizar una composición (b) o una recursión (c).

(a) Trivial . Las funciones base son totales por definición.

(b) Veamos la prueba para la composición.

Sea $f^{(n)}, g_1^{(m)}, g_2^{(m)}, \ldots, g_n^{(m)}$ funciones totales, queremos probar que

$$h^{(m)} = \Phi[f^{(n)}, g_1^{(m)}, g_2^{(m)}, \ldots, g_n^{(m)}]$$

es total.

Sea $X \in \mathbb{N}_0$.

Podemos calcular $x_i = g_i(X),\ i = 1 \ldots n$ pues cada g_i es total.

Entonces

$$\exists y \ \ y = f(x_1, x_2, \ldots, x_n)$$

pues f es total.

$$\therefore \exists y \in \mathbb{N}_0 \quad y = h(X).$$

(c) el caso de la recursión queda como ejercicio.

Esto nos permite afirmar que toda FRP es total. ■

No poder representar procesos de cálculo que estan definidos parcialmente sobre los naturales es una importante limitación de las **FRP**.

Pero, como mostraremos a continuacion, existen funciones calculables totales que no son **FRP**. Para mostrar un ejemplo de una función con tal característica debemos realizar una serie de pasos previos.

La idea es buscar una propiedad que cumplan todas las **FRP** y luego encontrar una función total y calculable que no la cumpla.

Comenzaremos presentando una serie de funciones.

3.2.2 La serie de Ackermann

Sea la siguiente sucesión de funciones que llamaremos serie de **Ackermann**

$$\begin{aligned} f_0(x) &= s(x) \\ f_1(x) &= f_0^{(x+2)}(x) = s^{(x+2)}(x) = 2x+2 \\ &\vdots \\ f_{k+1}(x) &= f_k^{(x+2)}(x) \\ &\vdots \end{aligned}$$

Veamos algunas propiedades de las funciones de esta serie.

Proposición 3.2.1.

I) $\forall k : f_k \in \boldsymbol{FRP}$

II) $x > x' \Rightarrow f_k(x) > f_k(x')$

III) $\forall x, k \Rightarrow f_k(x) > x$

IV) $\forall x, k \Rightarrow f_{k+1}(x) > f_k(x)$

Demostración.
Se demuestra por inducción sobre k utilizando la proposición 3.1.1, partiendo como caso base $k = 0$.

$$f_0(x) = s(x) \text{ que es } \mathbf{FRP} \text{ por ser base}$$

II) Ejercicio. (**Sugerencia:** hacer inducción sobre k)

III) Haremos inducción sobre k

Caso base $k = 0$

$$f_o(x) = s(x) > x$$

Paso inductivo

Suponemos que vale para $k = n$, o sea, tomamos como hipótesis $f_n(x) > x$ (**HI**), y probamos que es cierta que $f_{n+1}(x) > x$.

$$f_{n+1}(x) = f_n^{(x+2)}(x) = f_n(f_n^{(x+1)}(x) > f_n^{(x+1)}(x) = f_n(f_n^{(x)}(x)) > f_n^{(x)}(x) =$$

$$\vdots$$

$$= f_n(f_n^{(2)}(x)) \overset{\text{(HI)}}{>} f_n^{(2)}(x) = f_n(f_n(x)) \overset{\text{(HI)}}{>} f_n(x) \overset{\text{(HI)}}{>} x$$

IV)

$$f_{k+1}(x) \overset{(1)}{=} f_k^{(x+2)} \overset{(2)}{=} f_k^{(x+1)}(f_k(x)) \overset{\text{ii) y iii)}}{>} f_k^{(x+1)}(x) > \ldots > f_k(x)$$

(1) por definición de la $k+1$ función de la serie.
(2) por definición de potencia.

■

Definición 3.2.1. *Decimos que una función $f^{(1)}$ **mayora** a una $g^{(n)}$ si se verifica que*

$$f(max(x_1, x_2, \ldots, x_n)) \geq g(x_1, x_2, \ldots, x_n);\ \forall\ (x_1, x_2, \ldots, x_n)$$

Notaremos : $f^{(1)} \uparrow g^{(n)}$ *para indicar que* $f^{(1)}$ *mayora* $g^{(n)}$

Teorema 3.2.2. *Sea $g^{(n)} \in$ **FRP**, entonces existe f_k de la sucesión de Ackerman tal que:*

$$f_k \uparrow g$$

Demostración. Tal como hicimos para mostrar que todas las **FRP** son totales, haremos esta demostración en forma inductiva:

(a) Trivial. Las funciones base son mayoradas por f_0.

(b) Composición :
Sean las funciones:

$$I^{(m)}, h_1^{(n)}, h_2^{(n)}, \ldots, h_m^{(n)}$$

tales que

$$f_k \uparrow I^{(m)} \ \wedge f_k \uparrow h_i^{(n)} \quad i = 1, \ldots, m$$

.

$$\texttt{Sea} \quad g^{(n)} = \Phi[I^{(m)}, h_1^{(n)}, h_2^{(n)}, \ldots, h_m^{(n)}] \implies f_{k+1} \uparrow g^{(n)}$$

Por ser mayoradas por f_k se cumple que:

$$h_1^{(n)}(X) \leq f_k(max(X)),\ \ldots\ , h_m^{(n)}(X) \leq f_k(max(X))$$

Entonces

$$max\{h_1^{(n)}(X), h_2^{(n)}(X),\ \ldots\ , h_m^{(n)}(X)\} \leq f_k(max(X)) \quad (*)$$

Además como $f_k \uparrow I^{(m)}$

$$g(X) \stackrel{\text{def}}{=} I(h_1(X), h_2(X),\ \ldots\ , h_n(X)) \leq f_k(max(h_1(X), h_2(X),\ \ldots\ , h_n(X))) \leq$$

$$\stackrel{(1)}{\leq} f_k(f_k(max(X))) \stackrel{(2)}{\leq} f_k^{max(X)+2}(max(X)) \stackrel{\text{def}}{=} f_{k+1}(max(X))$$

$$\therefore f_{k+1} \uparrow g^{(n)}$$

(1) por $(*)$ y propiedad $ii)$.
(2) usando propiedades $ii)$ y $iii)$ de la proposición 3.2.1 aplicadas varias veces.

(c) Recursión
Sea $g^{(n+1)} = R[I^{(n)}, h^{(n+2)}]$, entonces

$$f_k \uparrow I \wedge f_k \uparrow h \Longrightarrow \ f_{k+1} \uparrow g^{(n)}$$

Para empezar, sabemos por definición de R que

$$g(0,X) = I(X)$$

$$g(y+1,X) = h(y,X,g(y,X))$$

pero $g(0,X) = I(X) \leq f_k(max(X))$ por hipótesis. $(**)$
Además,

$$g(1,X) = h(0,X,g(0,X)) \stackrel{\text{hip}}{\leq} f_k(max(0,X,g(0,X)) \leq$$

$$\stackrel{(1)}{\leq} f_k(max(0,X,f_k(max(X)) \stackrel{(2)}{\leq} f_k(f_k(max(X)))$$

Puede demostrarse por inducción que

$$\forall y,\ \ g(X,y) \leq f_k^{(y+1)}(max(X))$$

pero

$$f_k^{y+1}(max(X)) \stackrel{(3)}{\leq} f_k^{y+1}(max(y,X)) \stackrel{(4)}{\leq} f_k^{max(y,X)+1}(max(y,X)) \leq$$

$$\leq f_k^{max(y,X)+2}(max(y,X)) = f_{k+1}(max(y,X))$$

$$\therefore g(X,y) \leq f_{k+1}(max(y,X))$$

(1) por $ii)$ y $(**)$.
(2) $f_k(max(X)) > max(X,0) \quad \forall k$.
(3) agrego y al conjunto.
(4) por $ii)$.

■

A partir de (a), (b) y (c) podemos demostrar el teorema, ya que toda $f \in$ **FRP** se obtiene aplicando un número finito de veces los operadores Φ y R a las funciones base.

Teorema 3.2.3.

Definimos ahora una función que llamaremos ACK de la siguiente manera:

$$ACK(x) = f_x(x)$$

O sea para encontrar su valor imagen para un determinado x, tomamos la x-ésima función de Ackerman y la calculamos en dicho x.(existe un modo de calcularla para cualquier valor, aunque el esfuerzo de cálculo sea enorme)

*Mostraremos que esta función no pertenece al conjunto de las **FRP**.*

Demostración. Suponemos que $ACK \in$ **FRP**.

Luego, $ACK(x)+1$ debe pertenecer al conjunto de las **FRP**.

Si esta función fuese **FRP** debe existir, por lo visto en el teorema anterior, una función de la serie de Ackerman que la mayore.

Sea M tal que la M-ésima función de la serie, (f_M) mayora a $Ack(x)+1$ para todo valor de x

$$\forall x \quad ACK(x)+1 \leq f_M(x)$$

Tomemos entonces el caso en que $x = M$.

$$ACK(k)+1 \quad \leq f_M(M) \quad \text{pero como } f_M(M) = ACK(M)$$

$$ACK(k)+1 \quad \leq ACK(M)$$

Absurdo que proviene de suponer que ACK es **FRP**.

$$\therefore ACK \notin \textbf{FRP}.$$

■

Hemos encontrado una función que podemos calcular para todo número natural.

O sea que es total y calculable, pero no pertenece a las **FRP**.

3.3 Funciones Recursivas (FR)

Con la intención de completar nuestro modelo en esta sección agregaremos un nuevo operador:

3.3.1 Minimizador

Definición 3.3.1. *Dada $f^{(n+1)}$, decimos que $g^{(n)}$ se construye por minimización de f y lo notamos $M[f]$, cuando g es definida del modo siguiente:*

$$g^{(n)}(X) = M[f](X) = \mu_t(f(t,X) = 0)$$

es decir, para cada X^n el valor de $g(X)$ que se le asocia es (si existe) el mínimo t tal que $f(t,X) = 0$

Observemos que nada garantiza que exista tal valor de t , por lo que las funciones construidas con M pueden no estar definidas para todos los valores de la n-uplas.

Veamos algunos ejemplos de funciones sobre $\mathbb{N}_0^2$ que sólo están definidas para algunos pares de valores y que pueden construidas utilizando el operador M

Ejemplo 3.3.1. *La función cociente $C(x,y) = x/y$ que sólo está definida si x es múltiplo de y*

$$C(x,y) \equiv M\left[h(t,x,y)\right]$$

donde

$$h(t,x,y) \stackrel{def}{=} \left[(t.y)\dot{-}x\right] + \left[x\dot{-}(t.y)\right]$$

Ejemplo 3.3.2.
La función logaritmo $Log(x,y) = log_x y$ que sólo está definida si existe t tal que $x^t = y$

$$Log(x,y) \equiv M\left[h(t,x,y)\right]$$

donde

$$h(t,x,y) \stackrel{def}{=} \left[y\dot{-}x^t\right] + \left[x^t\dot{-}y\right]$$

Ejemplo 3.3.3.
La función raiz n-sima $Rad(x,n) = \sqrt[n]{x}$ que sólo está definida si existe t tal que $t^n = x$

$$Rad(x,n) \equiv M\left[h(t,x,n)\right]$$

donde

$$h(t,x,n) \stackrel{def}{=} \left[x\dot{-}t^n\right] + \left[t^n\dot{-}x\right]$$

Agregando este operador **Minimizador** a los anteriores definimos inductivamente un nuevo conjunto de funciones:

Definición 3.3.2. *Definiremos el conjunto de Funciones Recursivas de la siguiente manera:*

a) Las funciones base definidas en 3.1.5 son Funciones Recursivas.

b) Las funciones obtenidas a partir de Funciones Recursivas aplicándoles un número finito de veces los operadores composición, recursión y minimizador (Φ , R y M) son Funciones Recursivas.

*Las notaremos: **FR**.*

Observemos que las **FRP** son un subconjunto de las **FR**.

Podemos extender la propiedad de ser **recursivos** a los subconjuntos y a las relaciones definidas en $\mathbb{N}_0^k$

Definición 3.3.3.
Diremos que un subconjunto A de $\mathbb{N}_0^k$ es recursivo , *notaremos **CR** si su función característica $\chi_A : \mathbb{N}_0^k \to \{0,1\}$ es **FR***

Definición 3.3.4. *Dada una relación α definida en $\mathbb{N}_0^k$ diremos que es Relación Recursiva (notaremos **RR**) si su conjunto asociado es **CR***

Observación: Todos los mecanismos que permitian obtener **FRP** y **RRP** a partir de otras relaciones y funciones recursivas primitivas se pueden extender a las **FR** y **RR**.

Teorema 3.3.1. *Sea una relación R definida en $\mathbb{N}_0^{k+1}$*
Sea la función :

$$f(X) = \mu y(R(y,X^k))$$

*Enstonces si $R \in$ **RR** $\Longrightarrow$ $f \in$ **FR***

Demostración. Como $R \in \mathbf{RR}$ su función característica $\chi_{D_R} \in \mathbf{FR}$

$$f(X) = \mu\, y(R(y,X^k)) = \mu\, y\, (\chi_{D_R}(y,X) = 1) = \mu\, y\, (\chi_{D_{\neg R}}(y,X) = 0)$$

$$f = M\,[\chi_{D_{\neg R}}]$$

■

3.3.2 Funciones de Gödel

Sabemos que el conjunto de las listas finitas de números de $\mathbb{N}_0$ es numerable.

Esto implica que su cardinalidad es la misma de $\mathbb{N}_0$ y por lo tanto es posible encontrar funciones que asocien en forma unívoca un valor de $\mathbb{N}_0$ a cada lista finita de numeros naturales.

Definición 3.3.5. *Diremos que una función $G^{(2)}$ es función de **Gödel** si para cada una lista finita $< a_0, a_1, ...a_s >$ de elementos de $\mathbb{N}_0$, existe un valor z (al que se le llama número de Gödel de la lista) que tambien notaremos $z = \overline{< a_0, a_1, ...a_s >}$ tal que:*

I)

$$G(i,z) = a_i \quad i = 0,1,2,...s$$

II)

$$Se\ \ cumple\ \ que\ \ \overline{< x_0, x_1, ...x_s >} < \overline{< y_0, y_1, ...y_s, y_{s+1}, ...y_k >}$$

$$con\ \ \ s < k, \ \ x_i = y_i\ \ para\ \ i = 0, ...s$$

O sea que el número de Gödel asociado a cualquier lista finita es siempre menor que el que se le asocia al de la lista que se forma agregando nuevos elementos a la primera.

Ejemplo 3.3.4.

a) *Dada la sucesión de los números primos : $< 2,3,5,7,11,..... >$ podemos asociar en forma unívoca a cada sucesión finita de $\mathbb{N}_0$*

$$< a_0, a_1, ...a_k >$$

el valor

$$z = 2^{a_0}, 3^{a_1} .....N(k)^{a_k}$$

*Como la descomposición en factores primos es única esto nos garantiza poder reconstruir la sucesión original definiendo la siguiente función de **Gödel***

$$G(i,z) = \mu t(\neg(Mult(z, Exp(t+1, N_p(i)))))$$

Donde Mult, Exp y N_p fueron definidos en los ejemplos 3.1.10, 3.1.3 iv) y 3.1.14
*Observando que $G(i,z) \leq z$, aplicando el 3.1.3 se ve que esta función es **FRP***

b) *Sea una función **dupla** W y sus inversas U ,V*
A cualquier sucesión finita $< a_0, a_1, ...a_k >$ se le puede asociar unívocamente el número:

$$z = W(a_0, W(a_1, ...(W(a_k, 0))))$$

Entonces tenemos la función de Gödel:

$$G(i,z) = U(V^i(z))$$

*Utilizando las funciones dupla vistas en el ejemplo 3.1.15 se ve que esta función de Gödel también es **FRP***

3.3.3 Funciones Recorrido

Definición 3.3.6.

*Sea una función $f(y,X)$ y una función de **Gödel** $G^{(2)}$*

*Llamaremos función **recorrido** de f y la notaremos f^* a la función definida del siguiente modo:*

$$f^*(y,X) = \overline{< f(0,X), f(1,X), ...f(y,X) >}$$

Es decir a cada (y,X) se le asocia el valor que devuelve la $G^{(2)}$ sobre la y+1-upla formada por todos los valores $f(t,X)$ con t variando de 0 a y

$$G(i, f^*(y,X)) = f(i,X)$$

Teorema 3.3.2. *Si una $f \in FR \Longrightarrow f^* \in FR$*

En efecto $f^*(y,X)$ devuelve un número z tal que verifica la condición de Gödel para todo i de 0 a y

$$\bigcap_{i=0}^{y} G(i,z) = f(i,X)$$

Pero además es el menor número que cumple esa propiedad por la condición ii) de la definición de 3.3.5 Luego:

$$f^*(y,X) = \mu\, z \left[\bigcap_{i=0}^{y} G(i,z) = f(i,X) \right]$$

Luego $f^* \in FR$

3.3.4 Relaciones Semi-recursivas

Definición 3.3.7. *Dada una relación S definida en $\mathbb{N}_0^k$ diremos que es **Relación Semi-recursiva**, notaremos **RSR**, si existe una Relación R definida en $\mathbb{N}_0^{k+1}$, tal que $R \in$ **RR** y se cumple:*

$$S(X) \equiv \bigcup_{y} R(y,X)$$

Teorema 3.3.3.

$$\boldsymbol{RR} \subset \boldsymbol{RSR}$$

*Sea R definida $\mathbb{N}_0^k \in$ **RR**, la relación $S(y,X) = (y = 0) \wedge R(X)$ definida en $\mathbb{N}_0^{k+1}$ también pertenece a **RR***

Podemos presentar a R como:

$$R(X) \equiv \bigcup_{y} S(y,X) \quad \Longrightarrow R \in \boldsymbol{RSR}$$

Teorema 3.3.4. *Sea la S definida $\mathbb{N}_0^{k+1}$ y sea T la relación en $\mathbb{N}_0^k$ definida del siguiente modo:*

$$T(X) \equiv \bigcup_{z} S(z,X)$$

*Entonces si $S \in$ **RSR** $\Longrightarrow T \in$ **RSR***

*Por la definición de **RSR** existen una relación $R \in$ **RR** tal que:*

$$S(z,X) \equiv \bigcup_{y} R(y,z,X) \qquad T(X) \equiv \bigcup_{z} \bigcup_{y} R(y,z,X)$$

De donde:

$$T(X) \equiv \bigcup_{w} R(U(w), V(w), X)$$

Teorema 3.3.5.

Sean la S_1 , S_2 *relaciones definidas en* $\mathbb{N}_0^k$
Sean $A(X) \equiv S_1(X) \wedge S_2(X) \qquad B(X) \equiv S_1(X) \vee S_2(X)$
Entonces si S_1 , $S_2 \in$ ***RSR*** $\Longrightarrow \quad A, B \in$ ***RSR***
Si S_1 , $S_2 \in$ ***RSR*** *existen* R_1 , $R_2 \in$ ***RR*** *que les dan origen.*

$$A(X) \equiv \bigcup_{y} R_1(y,X) \wedge \bigcup_{z} R_2(z,X) \equiv \bigcup_{y}\bigcup_{z} (R_1(y,X) \wedge R_2(z,X)$$

$$B(X) \equiv \bigcup_{y} R_1(y,X) \vee \bigcup_{z} R_2(z,X) \equiv \bigcup_{y}\bigcup_{z} (R_1(y,X) \vee R_2(z,X)$$

Aplicando el teorema anterior $A, B \in$ ***RSR***.

Teorema 3.3.6. *(Este es el llamado* ***Lema de Post****)*

Sea la relación S *definida en* $\mathbb{N}_0^k$ *entonces*
S *y* $\neg S$ *son* ***RSR*** $\iff$ $S \in$ ***RR***

$$\Leftarrow)$$

Inmediato del teorema 3.3.3, y de la observación de la definición 3.3.4

$$\Rightarrow)$$

$$S(X) \equiv \bigcup_{y} R_1(y,X) \qquad \neg S(X) \equiv \bigcup_{y} R_2(y,X)$$

Obviamente $S(X) \vee \neg S(X)$ *es verdadera para todo valor de* X.

$$S(X) \vee \neg S(X) \equiv \bigcup_{y} R_1(y,X) \vee \bigcup_{y} R_2(y,X) \equiv \bigcup_{y} R_1(y,X) \vee R_2(y,X)$$

Sea $f(X) = \mu\ y(R_1(y,X) \vee R_1(y,X))$
La $f \in$ ***FR*** *por el teorema 3.3.1*
Entonces podemos ver que $S(X) \equiv R_1(f(X), X)$

3.3.5 Tesis de Church

Existe la presunción de que el conjunto de las **FR** coincide con el conjunto de las funciones "calculables", o seaaquellas funciones donde es posible obtener para cualquier conjunto de valores de su dominio, por medio de un mecanismo de cálculo, el valor de la imagen que corresponde a esos valores.

Cuando decimos "es posible" queremos indicar que existe un conjunto de instrucciones, que consisten en la acrítica aplicación de reglas establecidas rigurosamente. [1]

[1] Eso no implica que efectivamente se pueda realizar dicho calculo por ejemplo la función

$$A(m,n) \stackrel{\text{def}}{=} \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0, n = 0 \\ A(m-1, A(m,n-1)) & \text{si } m > 0, n > 0 \end{cases}$$

que es una variante de la Ack el valor de el resultado de A(5, 2) no se puede escribir como una secuencia de números pues su tamaño supera el estimado para el universo.

Considerar que coinciden la funciones "calculables" con las funciones recursivas es una afirmación conocida como la **Tesis de Church**

Observemos que no es posible demostrar lo que afirma la tesis de Church, dado que vincula un concepto perfectamente definido como el de las funciones recursivas con un concepto intuitivo como el de calculabilidad.

Sin embargo, da fuerza argumentativa a esta tesis que, en la búsqueda de otros posibles modelos para el cálculo, no se ha encontrado ningún mecanismo que no pueda ser representado con las **FR**

En los próximos capítulos veremos algunos de esos modelos.

3.4 Ejercicios

Ejercicio 3.4.1. Mostrar que para cada $n \in \mathbb{N}_0$, la función g_n tal que $g_n(x) = n$ para todo $x \in \mathbb{N}_0$ es recursiva primitiva.

Ejercicio 3.4.2. Mostrar que las siguientes son **FRP**:

1. $\Pi(x,y) = xy$
2. $exp(x,y) = x^y$
3. $fac(x) = x!$
4. La función *distancia*, definida por

$$dist(x,y) = \begin{cases} x-y & \text{si } x \geq y \\ y-x & \text{si } x < y \end{cases}$$

Ejercicio 3.4.3. Sea $f^{(k+1)}$ una **FRP**de orden $k+1$. Definimos dos nuevas funciones $F^{(k+1)}$ y $G^{(k+1)}$ de la siguiente manera:

$$F(X,y) = \sum_{k=0}^{y} f(X,k)$$
$$G(X,y) = \prod_{k=0}^{y} f(X,k)$$

donde X representa una k-upla. Mostrar que F y G son **FRP**

Ejercicio 3.4.4. Mostrar que todo subconjunto finito de $\mathbb{N}_0^k$ es un **CRP**

Ejercicio 3.4.5. Probar que si $A, B \subseteq \mathbb{N}_0^k$ son **CRP**, entonces $A \cup B$, $A \cap B$ y $\neg A$ son **CRP**.

Ejercicio 3.4.6. Mostrar que todos los subconjuntos finitos de $\mathbb{N}_0$ son **CRP**

Ejercicio 3.4.7. Mostrar que el conjunto de los números pares es un **CRP**

Ejercicio 3.4.8. Mostrar que el conjunto de los múltiplos de 3 es un **CRP**.

Sugerencia: probar que la función $r_3 : \mathbb{N}_0 \to \mathbb{N}_0$ que toma un natural y devuelve el resto que tiene en la división entera por 3 es una **FRP**y escribir la función característica de los múltiplos de 3 en términos de ésta.

Ejercicio 3.4.9. Mostrar que $=$, $\neq$, $\leq$ y $>$ son **RRP**

Ejercicio 3.4.10. Probar que la siguiente es una **FRP**:

$$f(x) = \begin{cases} x+3 & \text{si } x \text{ tiene resto 1 en la división por 3} \\ x! & \text{si } x \text{ tiene resto 2 en la división por 3} \end{cases}$$

Ejercicio 3.4.11. Probar que la relación de divisibilidad entre naturales es una **RRP**

Sugerencia: Defina la familia de funciones $r_a^{(1)}$, $a = 1,2,\ldots$; donde $r_a^{(1)}(n)$ devuelve el resto de dividir n por a, y escriba la función característica de la relación en términos de éstas funciones.

Ejercicio 3.4.12. Sea $R \subseteq \mathbb{N}_0 \times \mathbb{N}_0$ una relación recursiva primitiva. Defina la relación recursiva primitiva $T \subseteq \mathbb{N}_0 \times \mathbb{N}_0$, tal que

$$T(x,y) = \bigvee_{i=0}^{y} R(x,i)$$

Es decir, $T(x,y)$ es 1 si y sólo si $(R(x,0) = 1) \vee (R(x,1) = 1) \vee \cdots \vee (R(x,y) = 1)$

Ejercicio 3.4.13. Determine si el predicado

$$espot_2(x) = \begin{cases} 1 & \textit{si x es potencia de 2} \\ 0 & \textit{si no} \end{cases}$$

es recursivo primitivo.

Sugerencia: Utilice el ejercicio anterior.

Ejercicio 3.4.14. Es f una función recursiva primitiva?

$$f(x) = \begin{cases} x & \textit{si x es un número par mayor que 10} \\ 2x & \textit{si x es impar} \\ 3x & \textit{en otro caso} \end{cases}$$

Ejercicio 3.4.15. Demuestre que las funciones recursivas primitivas son totales.

Ejercicio 3.4.16. Sea $\{f_k \mid k \in \mathbb{N}_0\}$ el conjunto de funciones de Ackermann. Demuestre las siguientes propiedades:

1. $\forall k \in \mathbb{N}_0 : f_k$ *es FRP*
2. $\forall k \in \mathbb{N}_0 : f_k$ *es creciente*
3. $\forall k,x \in \mathbb{N}_0 : f_k(x) > x$

4 — Funciones Recursivas de Lista

4.1 Introducción

El modelo que veremos a continuación fue presentado por el extraordinario lógico **Giuseppe Jacopini** en cursos que dictara en la Universidad de Roma en los años 70.

Es un modelo poco conocido y una de las principales intenciones de este trabajo es difundirlo, ya que por su sencillez y elegancia resulta una muy didáctica aproximación a la problemática de modelizar el cálculo.

4.1.1 Listas finitas

Definición 4.1.1. *Una lista es una secuencia ordenada de* cero *o mas elementos pertenecientes a los* $\mathbb{N}_0$.

Cuando no estamos interesados en explicitar los elementos las notaremos con letras mayúsculas X,Y,Z.

Si queremos indicar que la lista tiene k elementos le agregaremos un índice: X^k

Si se desea explicitar sus elementos notaremos:

$$[x_1, x_2, ..., x_k]$$

En particular la lista de ***cero*** *elementos, la* ***lista vacía*** *la indicaremos:* [ ].

Definición 4.1.2.

LLamaremos $\mathscr{L}$ *al conjunto de todas las listas finitas con elementos de* $\mathbb{N}_0$

La ***longitud*** *de una lista es el número de elementos que posee.*

Para cada $m \in \mathbb{N}_0$ *notaremos con* $\mathscr{L}^m$ *el conjunto de las listas que poseen exactamente m elementos.*

Notaremos $\mathscr{L}^{\geq m}$ *el conjunto de las listas que poseen al menos m elementos.*

La operación natural en $\mathscr{L}$ *es la* ***concatenación***

Dadas dos listas $X = [x_1, x_2, ..., x_m]$ $Y = [y_1, y_2, ..., y_k]$ *llamamos* ***concatenación*** *de X y Y la lista:*

$$[x_1, x_2, ..., x_m, y_1, y_2, ..., y_k]$$

Es sencillo mostrar que esta es una operación asociativa que tiene como elemento neutro la lista vacía

En algunos casos notaremos X, Y *o* $[X, Y]$ *para indicar la lista obtenida al concatenar la lista X y la lista Y*

Tambien usaremos, para distinguir en particular algunos elementos, notaciones del tipo a,X, $,b$, Y *o* $[a,X, b, Y]$ *para referirnos, por ejemplo, a la lista que se obtiene de concatenar las listas* $[a]$, X, $[b]$ *e* Y.

4.2 Funciones de listas

Podemos definir ahora las funciones de listas como funciones que van de $\mathscr{L}$ a $\mathscr{L}$

Las notaremos con letras mayúsculas: F, G.

Observemos que las funciones numericas $f : \mathbb{N}_0^n \to \mathbb{N}_0$ definidas en el capítulo anterior, se pueden considerar un acaso particular de las funciones de lista en las que el dominio es $\mathscr{L}^k$ y el codominio $\mathscr{L}^1$.

Dada una función de lista F indicaremos el dominio de la función como $\mathscr{D}_F$

En muchos casos para indicar como opera una función F sobre una lista dada notaremos $F(X) = Y$ como $X \xrightarrow{F} Y$ o tambien $X \overset{F}{\text{---}} Y$

A continuación, tal como hicimos para las funciones numericas, definiremos un conjunto de funciones base y operadores que nos permitan construir recursivamente un conjunto particular de funciones de lista.

Definición 4.2.1 (Funciones base)**.** *En este caso nos alcanza con sólo seis funciones base que son las siguientes:*

I) ***Cero a izquierda***
Esta función agrega un 0 *en el extremo izquierdo de la lista.*
La notaremos: 0_i

$$0_i : \mathscr{L} \to \mathscr{L} \qquad 0_i\,[x_1, x_2, ..., x_k] \longmapsto [0, x_1, x_2, ..., x_k]$$

$$X \xrightarrow{0_i} 0, X$$

II) ***Cero a derecha***
Esta función agrega un 0 *en el extremo derecho de la lista.*
La notaremos: O_d

$$O_d : \mathscr{L} \to \mathscr{L} \qquad 0_d\,[x_1, x_2, ..., x_k] \longmapsto [x_1, x_2, ..., x_k, 0]$$

$$X \xrightarrow{0_d} X, 0$$

III) ***Borrar a izquierda***
Esta función, cuyo dominio son las listas no vacías, borra el elemento del extremo izquierdo de la lista.
La notaremos: $\square_i$

$$\square_i : \mathscr{L}^{>0} \to \mathscr{L} \qquad \square_i\,[x_1, x_2, ..., x_k] \longmapsto [x_2, ..., x_k]$$

$$y, X \xrightarrow{\square_i} X$$

IV) ***Borrar a derecha***
Esta función, , cuyo dominio son las listas no vacías, borra el elemento del extremo derecho de la lista.
La notaremos: $\square_d$

$$\square_d : \mathscr{L}^{>0} \to \mathscr{L} \qquad \square_d\,[x_1, x_2, ..., x_{k-1}, x_k] \longmapsto [x_1, x_2, ..., x_{k-1}]$$

$$X, y \xrightarrow{\square_d} X$$

V) ***Sucesor a izquierda***
Esta función, , cuyo dominio son las listas no vacías, incrementa en uno el elemento del extremo izquierdo de la lista.
La notaremos: S_i

$$S_i : \mathscr{L}^{>0} \to \mathscr{L} \qquad S_i[x_1, x_2, ..., , x_k] \longmapsto [(x_1) + 1, x_2, ..., x_k]$$

$$y, X \xrightarrow{S_i} y + 1, X$$

VI) ***Sucesor a derecha***
Esta función, , cuyo dominio son las listas no vacías, incrementa en uno el elemento del extremo izquierdo de la lista.
La notaremos: S_d

$$S_d : \mathscr{L}^{>0} \to \mathscr{L} \qquad S_d[x_1, x_2, ..., , x_k] \longmapsto [x_1, x_2, ..., x_k + 1]$$

$$X.y \xrightarrow{S_d} X, y + 1$$

Ejemplo 4.2.1.

1.

$$O_i[3,4,9] = [0,3,4,9] \quad ; \quad O_d[3,4,9] = [3,4,9,0]$$

2.

$$\square_i[3,4,9] = [4,9] \quad ; \quad \square_d[3,4,9] = [3,4]$$

3.

$$S_i[3,4,9] = [4,4,9] \quad ; S_d[3,4,9] = [3,4,10]$$

4.2.1 Operadores

Sólo son necesarios dos operadores para la construcción recursiva del de conjunto nuevas funciones.

Definición 4.2.2 (Operador Composición)**.**
Sean F, G dos funciones de lista.

$$F : \mathscr{D}_F \longrightarrow \mathscr{L} \qquad G : \mathscr{D}_G \longrightarrow \mathscr{L}$$

*A partir de las mismas decimos que construimos por **composición** una nueva función de lista H (notaremos $H = FG$)*
$X \in \mathscr{D}_{FG}$ si $X \in \mathscr{D}_F$ y $F[X] \in \mathscr{D}_G$, entonces $H[X] = G[F[X]]$

O sea que consiste en operar primero la función F y al resultado aplicarle la función G

Definición 4.2.3 (Operador Repetición)**.** *Sea F una función de lista. Diremos que a partir de la misma se cosntruye por **repetición**una nueva función H, que notaremos $H = \langle F \rangle$ definida del siguiente modo:*
Sea una lista de la forma x, X, y (o sea la presentamos de modo tal de distinguir el primero y el último elemento de la lista .(observar que la parte que denominamos con X eventualmente puede ser la lista vacía.

$H = \langle F \rangle$ opera del siguiente modo:

$$H = \langle F \rangle [x,X,y] = \begin{cases} x,X,y & \text{si } x = y \\ \langle F [x,X,y] \rangle & \text{si x} \neq \text{y} \end{cases}$$

O sea $\langle F \rangle$ consiste en repetir la aplicación de F hasta que el primer componente de la n-upla dato sea igual al último. Por lo tanto $X \in \mathscr{D}_{\langle F \rangle}$ si en la sucesión $X, F[X], F[F[X]], \ldots$ hay un elemento en el que primero y último elemento son iguales.

Veamos algunos ejemplos de la aplicación de estos operadores:

Ejemplo 4.2.2. *1.* ***Pasar a la izquierda***
Esta función pasa el elemento que está en extremo derecho de la lista al extremo izquierdo .
La notaremos: $\lhd$

$$X, y \xrightarrow{\lhd} y, X$$

Se puede ver que se construye a partir de las funciones base y los operadores del siguiente modo:

$$\lhd \; = O_i \langle S_i \rangle \square_d$$

2. ***Pasar a la derecha***
Esta función, similar a la anterior, pasa el elemento que está en extremo izquierdo de la lista al otro extremo .
La notaremos: $\rhd$

$$y, X \xrightarrow{\rhd} X, y$$

Se puede construir del siguiente modo:

$$\rhd \; = O_d \langle S_d \rangle \square_i$$

3. ***Duplicar a izquierda***
Esta función, que tiene como dominio las listas de al menos un elemento, duplica el primer elemento de la lista.
La notaremos: D_i

$$D_i \;\; [x,Y] \longrightarrow [x,x,Y]$$
$$x, Y \xrightarrow{D_i} x, x, Y$$

Se construye del siguiente modo:

$$D_i \; = \; 0_d \; \langle S_d \rangle \; \lhd$$

4. ***Duplicar a derecha***
Similar a la anterior agrega al final de la lista una copia del último elemento de la lista.
La notaremos: D_d

$$D_d \;\; [Y,x] \longrightarrow [Y,x,x]$$
$$Y, x \xrightarrow{D_d} Y, x, x$$

Se construye del siguiente modo:

$$D_d \; = \; 0_i \; \langle S_i \rangle \; \rhd$$

5. ***Cambia extremos***
   Esta función intercambia los extremos de la lista.
   La notaremos: $\leftrightarrow$

$$\leftrightarrow \ [x,X,y] \longrightarrow [y,X,x]$$

$$x,X,y \xrightarrow{\leftrightarrow} y,X,x$$

Se sugiere tratar de construir esta función y luego controlar con la propuesta que sigue:

$$\leftrightarrow \ = \ \rhd 0_i \lhd 0_i \langle S_i \rhd \rhd S_i \lhd \lhd \rangle \square_d \square_i$$

A partir de las seis funciones base y los dos operadores construiremos en forma inductiva el conjunto de las **Frunciones Recursivas de Lista**

Definición 4.2.4.

Definiremos el conjunto de Funciones Recursivas de Lista de la siguiente manera:
a) Las funciones base definidas en 4.2.1 son Funciones Recursivas de Lista.
b) Las funciones obtenidas a partir de Funciones Recursivas de Lista aplicándoles un número finito de las operaciones de composición y repetición definidas en 4.2.2 y 4.2.3 son Funciones Recursivas de Lista

*Las notaremos **FRL**.*

*En particular todas las funciones vistas en el ejemplo anterior son **FRL***

*Podriamos, tal como lo hicimos con las **FR** ir mostrando diversos cálculos que pueden ser representados por las **FRL***

*Pero, es mas conveniente mostrar directamente que las **FRL** son, al menos, tan poderosas como las **FR** para representar procedimientos de cálculo.*

4.2.2 El poder de cálculo de las FRL

Teorema 4.2.1.

*Para toda función $f^k \in$ **FR** existe una función en **FRL** , que notaremos F_f tal que:*
Dada la lista X^k que está en el dominio de f y cualquier lista Y

$$F_f\,[X,Y] \longmapsto f(X),\ X,\ Y \quad X,Y \in \mathscr{D}_{F_f} \iff X \in \mathscr{D}_f$$

$$X,\ Y \xmapsto{F_f} f(X),\ X,\ Y$$

Diremos que F_f "representa " a f

Mostraremos este teorema en forma inductiva.

Lema 4.2.1.

*Las funciones base de las **FR** son representadas por **FRL***

1. *Las funciones cero c^k son, para cualquier k, representadas por $F_c = 0_i$ En efecto:*

$$X,\ Y \xmapsto{F_c} 0,\ X,\ Y$$

2. *Las proyecciones $p_i^{(n)}$, son representadas por F_{p_i} que se construye de la siguiente manera :*

$$F_{p_i} = \underbrace{\rhd \rhd \ldots}_{i\text{ - }1\text{ veces}} D_i \ \underbrace{\leftrightarrow \lhd \leftrightarrow \lhd \ldots}_{i\text{ - }1\text{ veces}}$$

Observar que estos representantes para las proyecciones $p_i^{(n)}$ sólo dependen del valor de i .

3. *El sucesor es representado por $F_s = D_i\ S_i$*

*Luego todas las funciones base de **FR** se pueden representar con **FRL***

Lema 4.2.2. *La composición de funciones que tienen representación en **FRL**, tiene representación en **FRL***

*Sea la función numérica $f^{(n)}$ y una familia de n funciones numéricas de orden k , $\{g_i^{(k)}\}_{i=1}^n$ tales que tienen representación en **FRL** : $F_f, F_{g_1}, F_{g_2}, ..., F_{g_n}$ entonces*

$$h = \Phi\left(f^{(n)}, g_1^{(k)}, g_2^{(k)}, ..., g_n^{(k)}\right)$$

*tiene representación en **FRL***

Se puede ver que :

$$F_h = F_{g_1} \rhd \ F_{g_2} \rhd \ \ldots \ F_{g_n} \underbrace{\lhd \lhd \ldots}_{n\text{ - }1\text{ veces}} F_f \rhd \underbrace{\square_i \square_i \ldots}_{n\text{ veces}} \lhd$$

En efecto:

$$X,Y \ \xrightarrow{F_{g_1} \rhd \ F_{g_2} \rhd \ \ldots \ F_{g_n} \lhd\lhd \ldots} \ g_1(X),\ g_2(X),\ldots g_n(X),X,Y$$

$$g_1(X),g_2(X),\ldots g_n(X),X,Y \xrightarrow{F_f \rhd \ \square_i \square_i \ldots \lhd} \ f(X),X,Y$$

Lema 4.2.3. *La función construida por recursión de funciones numéricas que tienen representación en las **FRL** tiene representación en las **FRL***

*Sean las funciones numéricas $g^{(k)}$ y $h^{(k+2)}$ tales que existen F_g y F_h que las representan en las **FRL** entonces $f^{(k+1)} = R(g^{(k)}, h^{(k+2)})$ es representable en **FRL***

Se puede mostrar que :

$$F_f = \rhd F_g\ 0_i \langle \ \rhd\leftrightarrow\lhd F_h \rhd \ \square_i\ S_i \ \leftrightarrow \lhd \ \rangle \ \square_i \ \leftrightarrow \lhd$$

Veamos los pasos:

$$y,X,Y \ \xrightarrow{\rhd F_g\ 0_i} \ 0,\ g(X),\ X,\ Y,\ y$$

Si $y = 0$ entonces, por ser iguales los extremos no se aplica la función entre $\langle\rangle$ y directamente se pasa a la parte que sigue:

$$0,\ g(X),\ X,\ Y,\ 0 \ \xrightarrow{\square_i \ \leftrightarrow \lhd} \ g(X),\ 0,\ X,\ Y$$

Si no son iguales los extremos se aplica la repetición de la la función del operador $\langle\rangle$

$$t, f(t,X),\ X,\ Y,\ y \ \xrightarrow{\rhd\leftrightarrow\lhd F_h \rhd \ \square_i\ S_i \ \leftrightarrow \lhd} \ t+1,\ f(t+1,X),\ X,\ Y,\ y$$

Esto se repite hasta que los extremos sean iguales:

$$y, f(y,X),\ X,\ Y,\ y \quad \frac{\Box_i \leftrightarrow \lhd}{} \quad f(y,X),\ y,\ X,\ Y$$

Lema 4.2.4.

La función numérica que se obtiene por minimización de una función representable en ***FRL*** *tiene representación en* ***FRL****.*

Sea $g^{(n+1)}$ que tiene representación F_g en ***FRL****, entonces $f^{(n)} = M[g]$ es representable.*

Se puede verificar que:

$$F_f \ = \ 0_i\ F_g\ 0_d\ \langle \Box_i\ S_i\ F_g\ \rangle \Box_i\ \Box_d$$

$$X,\ Y \ = \ \frac{0_i\ F_g\ 0_d}{}\ g(0,X),\ 0,\ X,\ Y,\ 0$$

Si $g(0,X) = 0\ 0$ es el valor buscado, no se aplica la parte entre $\langle\rangle$ y queda:

$$g(0,X),\ 0,\ X,\ Y,\ 0 \ = \ \frac{\Box_i\ \Box_d}{}\ 0\ ,\ X,\ Y \ = \ f(X),X,\ Y$$

Si no son iguales los extremos se repite la función entre $\langle\rangle$ hasta que esto suceda:

$$g(t,X),\ t,\ X,\ Y,\ 0 \ = \ \frac{\Box_i\ S_i\ F_g}{}\ g(t+1)\ ,t+1,\ X,\ Y,0$$

Repito hasta que $g(k,X)$ sea igual a 0

$$g(k,X),\ k,\ X,\ Y,\ 0 \ = \ \frac{\Box_i\ \Box_d}{}\ k\ ,\ X,\ Y \ = \ f(X),X,\ Y$$

Como por definición las ***FR*** *se construyen a partir de las funciones base aplicando los operadores Φ , R y M , de los lemas vistos se concluye que toda* ***FR*** *tiene una repesentación en las* ***FRL***

Es entonces que podemos afirmar que las ***FRL*** *son un modelo del cálculo tan poderoso como el de las* ***FR****.*

Observando que, adicionalmente, esta representación tiene la característica de conservar los datos, posemoa llegar a pensar que tal vez las ***FRL*** *sean mas potentes como modelo, permitiendo representar cálculos que no pueden representar las* ***FR*** *(rebatiendo la tesis de Church). En el próximo capítulo veremos que eso no sucede.*

4.3 Ejercicios

Ejercicio 4.3.1. Calcular.

(a) $S_d S_d S_d \Box_d \Box_d\,[3,4,5]$
(b) $\rhd \leftrightarrow \lhd\,[1,2,3,4,5]$
(c) $\langle \Box_d \Box_i \rangle\,[1,3,5,7,9,2,3,7,5,11,13]$
(d) $\langle S_d \rangle\, O_d\,[5,X]$
(e) $\langle \rhd\, O_d \lhd S_d \rangle\, O_d\,[4,X]$
(f) $\langle S_i \leftrightarrow S_d \rangle\,[1,X,1]$
(g) $\langle S_d S_d \rangle\, O_d\,[4,3,2,1]$
(h) $\lhd \Box_i \Box_d \langle \rhd\, S_d \lhd S_d \rangle\, O_d \rhd\,[2,3,10,11]$

Construir en cada caso, a partir de las funciones elementales, la composición y la repetición, una función f tal que

(a) $f[X] = [7, X]$
(b) $f[x, Y] = [x+5, Y]$
(c) $f[x, Y] = [x, x, Y]$
(d) $f[x, Y] = [x, x, x, Y]$
(e) $f[x, Y, z] = [x, z, Y]$
(f) $f[x, Y] = [x, x+1, x+2, Y]$
(g) $f[x, y, z, W] = [z, y, x, W]$

Ejercicio 4.3.2. Sea $f = \langle S_i S_i S_i \rangle O_i$ Cuáles de las siguientes listas pertenecen al dominio de f?:
(I) $[1, 2, 3]$
(II) $[]$
(III) $[3]$
(IV) $[5]$
(V) $[6, 5, 4, 3, 2, 1]$
(VI) $[1, 2, 3, 4, 5, 6]$
Exprese por comprensión el conjunto $dom(f)$.

Ejercicio 4.3.3. Cuál es el dominio de las siguientes funciones de listas?
1. $S_i \square_d \square_i S_i$
2. $S_d \square_d O_i$
3. $S_d \square_d \square_d \square_d O_i$
4. $\langle S_d S_d \rangle O_d$
5. $\langle S_i \rangle O_i O_d$
6. $\langle S_i \rangle S_i O_i O_d$
7. $\langle S_d O_d \rangle S_i S_i S_i S_i O_i$

Ejercicio 4.3.4. Definir las siguientes funciones de listas. En cada caso, explicite el dominio.
(a) $\pi[x, y, Z] = [x.y, Z]$
(b) $f[x, y, Z] = [x^y, x, y, Z]$
(c) $pot[x, Y] = \left[1 + 2x + x^2, Y\right]$. Observación: Asumiremos $x^y = 1$ para el caso $x = y = 0$
(d) $g[x, Y] = \begin{cases} \left[\frac{x}{2}, Y\right] & \text{si } x \text{ es par} \\ indefinida & \text{en caso contrario} \end{cases}$

Ejercicio 4.3.5. Dé una expresión para las siguientes funciones de listas:
1.
$$\begin{array}{rcl} pred \;:\; \left\{[x, Y] \in \mathscr{L}^{\geq 1} \mid x \geq 1\right\} & \to & \mathscr{L} \\ [x, Y] & \mapsto & pred[x, Y] = [x-1, Y] \end{array}$$
2.
$$\begin{array}{rcl} resta \;:\; \left\{[x, y, Z] \in \mathscr{L}^{\geq 2} \mid x \geq y\right\} & \to & \mathscr{L} \\ [x, y, Z] & \mapsto & resta[x, y, Z] = [x-y, Z] \end{array}$$

Ejercicio 4.3.6. Defina funciones de listas cuyos dominios sean los siguientes:
(a) $\mathscr{L}^{\geq 4}$
(b) $\left\{[x_1, x_2, \ldots, x_n] \in \mathscr{L}^{\geq 2} \mid x_2 = 0\right\}$
(c) $\left\{[x_1, \ldots, x_n] \in \mathscr{L}^{\geq 2} \mid x_1 = x_n + 1\right\}$
(d) $\left\{[x_1, x_2, \ldots, x_n] \in \mathscr{L}^{\geq 3} \mid x_1 + x_2 = x_n\right\}$

Ejercicio 4.3.7. Construya, en cada caso, una función f tal que
(I) $f[x, Y] = [0, 1, 2, \ldots, x, Y]$
(II) $f[x, Y] = \left[x^3, Y\right]$
(III) $f[x, y, Z] = \langle Z, \underbrace{y, y, \ldots, y}_{\text{x veces}} \rangle$

(IV) $f\,[x,y,Z] = \left[(x+y)^2, Z\right]$
(V) $f\,[a,b,c,n,X] = [a^n, b^n, c^n, a, b, c, n, X]$
(VI) $f\,[x,Y] = [1+2+\cdots+x, X]$

Ejercicio 4.3.8. Defina una función de listas f tal que

$$f\,[a,b,c,X] = \begin{cases} [a,b,c,X] & \textit{si } a.b = 1+2\cdots+c \\ \text{indefinida} & \textit{en otro caso} \end{cases}$$

5 — Máquina de Turing

5.1 Introducción

En este capítulo veremos un camino diferente para encontrar una modelización del cálculo. .

La idea base es suponer que todo cálculo puede ser realizado "mecanicamente " por un autómata que aplica rigurosa y acriticamente un conjunto de reglas ante determinadas situaciones y fue desarrollada por el genial lógico **Alan Turing**.

5.2 Descripción de la Máquina de Turing

Existen diversas descripciones, todas en esencia equivalentes, del autómata creado por Turing, al que notaremos genéricamente **MT**

La información que procesa esta máquina ideal se encuentra en una cinta dividida en casillas en las que hay símbolos de un alfabeto finito:

$$A = \{s_0, s_1, s_2, \ldots s_n\}$$

Este alfabeto tiene al menos dos símbolos.

Uno de tales símbolos, que notaremos con s_0 se usa para indicar que la casilla está vacía. A veces utilizaremos el nombre "blanco" para referirnos al mismo, y también lo notaremos con $\square$

La cinta se considera infinita, pero se conviene que sólo un conjunto finito de casillas tienen, al iniciar el proceso, símbolos diferentes de s_0 (es decir no están vacías).

La **MT** en cada momento se considera que está "observando " o "leyendo" una casilla (se puede pensar en un visor que se desliza por la cinta) y se encuentra en un "estado" q_j que pertenece a un conjunto finito de estados Q.

$$Q = \{q_0, q_1, q_2, \ldots q_m\}$$

El conjunto de estados tiene siempre, al menos, dos estados: uno llamado "estado inicial". que es en el que se encuentra la **MT** al comenzar el cálculo (por convención lo codificaremos con q_1) y otro llamado "estado final" (convenimos en codificarlo q_0) en el cual la máquina se detiene y da por terminado el cálculo.

Dependiendo del estado en que se encuentra (supongamos q_r) y del símbolo que esta en la casilla observada (supongamos s_i) la **MT** realiza consecutivamente tres operaciones:

a) Símbolo: Sustituye el símbolo observado s_i por otro pertenenciente al alfabeto A. (eventualmente puede ser el mismo símbolo)
b) Movimiento: Se mueve a la celda derecha o a la celda izquierda o permanece en la misma celda. (notaremos d, i, n las tres alternativas)
c) Estado : Sustituye el estado en que se encuentra q_r por otro pertenenciente al conjunto de estados Q.(eventualmente puede ser el mismo estado)

Partiendo de una configuración dada de la cinta (que llamamos : *configuración inicial*) y del estado inicial (q_1), se repiten los pasos vistos hasta que la **MT** alcanza el que llamamos "estado final"(q_0) y se detiene. (lo que no siempre sucede necesariamente).

Si se llega al estado q_0, la configuración de la cinta en ese estado (se le llama *configuración final*) es el reultado del cálculo.

Por lo tanto una **MT** queda totalmente determinada por su alfabeto, sus estados y una función que le indica que hacer en cada momento a partir del estado en que se encuentra y el símbolo que está observando.

$$f_{TM} : A \times Q - \{q_0\} \longrightarrow A \times Q \times M$$

Donde con A representamos el alfabeto, $Q - \{q_0\}$ los estados distintos del final y M es el conjunto de los posibles movimientos $M = \{d, i, n\}$

Esta función, de dominio finito, puede ser representada por una tabla de $(n+1)$ filas correspondientes a cada uno de los elementos del alfabeto y m columnas correspondientes a los estados q_1 a q_m-

En cada una de las casillas i, j , asociadas al símbolo s_i del alfabeto y al estado q_j aparecerá una terna s_m, m_k, q_h que indicará respectivamente el símbolo a escribir, el movimiento a ejecutar y el estado sucesivo que corresponda.

La primera columna de la tabla (correspondiente al estado q_1) es la del estado en que se encuentra la máquina en el momento de comenzar a trabajar.

Ejemplo 5.2.1. *Tomando como alfabeto* $A = \{\square, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

La máquina que tiene esta tabla

	q_1	q_2
$\square$	$\square, i, q_2$	$1, n, q_0$
0	$0, d, q_1$	$1, n, q_0$
1	$1, d, q_1$	$2, n, q_0$
2	$2, d, q_1$	$3, n, q_0$
3	$3, d, q_1$	$4, n, q_0$
4	$4, d, q_1$	$5, n, q_0$
5	$5, d, q_1$	$6, n, q_0$
6	$6, d, q_1$	$7, n, q_0$
7	$7, d, q_1$	$8, n, q_0$
8	$8, d, q_1$	$9, n, q_0$
9	$9, d, q_1$	$0, i, q_2$

Si en la configuración inicial en la cinta hay escrito un número (base 10) y la casilla observada es el primer dígito , al final del proceso en la cinta aparece el numero siguiente.

Ejemplo 5.2.2. *Tomando como alfabeto* $A = \{\square, \bullet\}$

La máquina que tiene esta tabla

	q_1	q_2	q_3	q_4	q_5
$\square$	$\square, n, q_0$	$\square, d, q_3$	$\bullet, i, q_4$	$\square, i, q_5$	$\bullet, d, q_1$
$\bullet$	$\square, d, q_2$	$\bullet, d, q_2$	$\bullet, d, q_3$	$\bullet, i, q_4$	$\bullet, i, q_5$

En el estado inicial en la cinta hay escrito una cantidad de "• " y la casilla observada es la del primer •, al final quedan dos grupos iguales al inicial separados por un blanco.

5.3 Composición de Máquinas de Turing

Dadas dos máquinas M_1 y M_2, *ambas con el mismo alfabeto*, llamamos **composición** de M_1 con M_2 (Notamos: $M_1\ M_2$) a la máquina que realiza la operación equivalente a aplicar primero la máquina M_1 y al resultado aplicar la M_2.

Es sencillo construir, a partir de las tablas de las máqunas M_1 y M_2 la tabla de la $M_1\ M_2$.

Para ello se adjunta inmediatamente a la tabla de M_1, la tabla de M_2 cambiando los nombres de los estados para que no se confundan (por ej. denominando cada estado q_i de M_2 , $q_i{}'$) y reemplazando en la tabla de la M_1 cada aparición del estado final (q_0) con el inicial de la máquina M_2 , ($q_1{}'$).

	q_1	q_2	q_3	q_4
s_0	...		...	..
s_1	...		$..q_0$	..
s_2	...		...	$..q_0$

M_1

	q_1	q_2	q_3
s_0	...	...	...
s_1	...	...	...
s_2	...	...	...

M_2

	q_1	q_2	q_3	q_4	$q_1{}'$	$q_2{}'$	$q_3{}'$
s_0	...	...	...	...	...	...	...
s_1	...	$q_1{}'$	...	...	...	...	...
s_2	...	...	$q_1{}'$	...	...	...	...

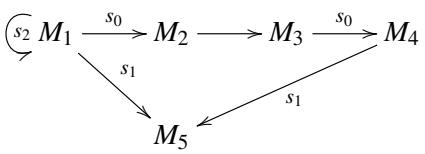
M_1M_2

5.4 Diagramas de composición

Veamos una forma mas general de componer máquinas. En este caso ante el estado final de una máquina se toma en consideración el símbolo observado, y, dependiendo del mismo, se la conecta con el estado inicial de una determinada máquina.(eventualmente la misma)

Esto se diagrama con una flecha, marcada con el símbolo que parte de la máquina que ha terminado hacia la que continua con el proceso. Si no se etiqueta la flecha de salida implica que en todos los casos se pasa al estado inicial de la máquina a la que apunta la flecha.

Supongamos varias máquinas: M_1,M_2, M_3,M_4,M_5, todas sobre el mismo alfabeto (en este ejemplo $[s_0, s_1, s_2]$).



Este diagrama indica que si M_1 alcanza el estado q_0, observando el símbolo s_2 vuelve a su estado q_1, si termina con s_0 va al estado q_1 de M_2 y si finaliza con s_1 al estado q_1 de M_5.

La tabla de la máquina compuesta se construirá renombrando los estados de cada máquina para evitar coincidencias, agrupando las tablas de las máquinas y en cada tabla donde aparece el estado q_0, reemplazandolo, si fuera necesario, de acuerdo al símbolo del alfabeto, con el estado ${q_1}^j$ de la máquina j que le corresponde.

5.5 Máquinas elementales

Se puede mostrar que con un alfabeto de dos símbolos podemos codificar cualquier alfabeto finito. Por lo tanto,de ahora en adelante, salvo aclaración, trabajaremos con **MT** que tendrán como alfabeto $A = \{\square, \bullet\}$.

Consideremos las siguientes cinco máquinas elementales:

1. Máquina "Mueve a derecha" (la notaremos: **d**)

	q_1
$\square$	$\square, d, q_0$
$\bullet$	$\bullet, d, q_0$

2. Máquina "Mueve a izquierda" (la notaremos: **i**)

	q_1
$\square$	$\square, i, q_0$
$\bullet$	$\bullet, i, q_0$

3. Máquina "Blanco" (la notaremos: $\square$)

	q_1
$\square$	$\square, n, q_0$
$\bullet$	$\square, n, q_0$

4. Máquina "Punto" (la notaremos: $\bullet$)

	q_1
$\square$	$\bullet, n, q_0$
$\bullet$	$\bullet, n, q_0$

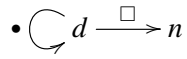
5. Máquina "Nada" (la notaremos: **n**)

	q_1
$\square$	$\square, n, q_0$
$\bullet$	$\bullet, n, q_0$

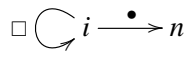
Las dos primeras se mueven a la derecha o a la izquierda sin modificar el contenido de la celda observada. Las otras dos sustituyen el contenido de la celda observada incondicionalmente por el símbolo de su nombre. Finalmente la última, como su nombre lo indica, no hace nada.

A partir de estas máquinas elementales se puede construir cualquier **MT** sobre el alfabeto dado.

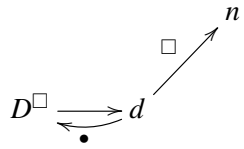
Ejemplo 5.5.1. *Sea $D^{\square}$ la máquina que avnza por la cinta hacia la derecha hasta que encuentra el primer $\square$ Se puede componer de la siguiente manera:*



Analogamente podemos definir $I^{\bullet}$, una máquina que avanza hacia a la izquierda hasta encontrar el primer $\bullet$, por medio del siguiente diagrama:



La máquina $D^{\square\square}$ que avanza hacia la derecha hasta que encuentra dos casillas vacías es la siguiente:



En forma similar podemos definir las máquinas $I^{\square}$, $I^{\bullet\square}$, etc.

5.6 Poder de cálculo de las MT

Mostraremos que el poder de calculo de las **MT** es, al menos tan poderoso, como el de las **FR**.

Para ello, como elemento previo debemos demostrar la existencia de una particular **MT** que llamaremos la máquina **Z**

5.6.1 La Máquina Z

La máquina **Z** es una **MT** que realiza la siguiente tarea:

Como dato de entrada se le debe presentar un conjunto de al menos dos grupos de • separados por un solo □, con todo el resto de la cinta en blanco.

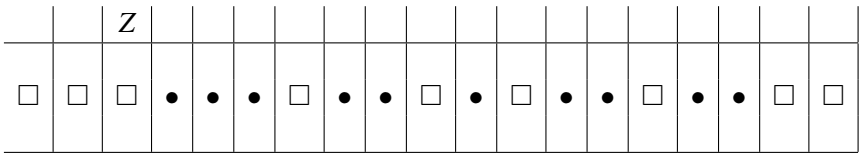
La máquina *Z* parte en la primera celda vacía que está a la izquierda del primer grupo.

La configuracion final de la cinta es la misma que la inicial, pero la celda observada difiere de acuerdo a lo siguiente:

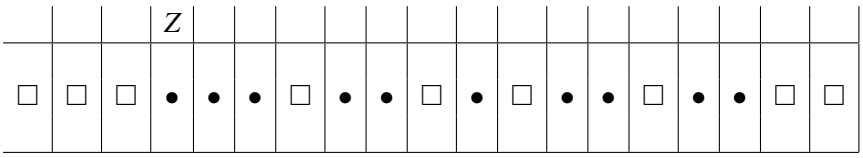
Si el primer grupo de • tiene el mismo número de elementos que el último grupo entonces la casilla en que se para la máquina es la misma en la que partió (queda en la primera vacía a la izquierda).

Si los grupos primero y último no tienen el mismo número de elementos, entonces la casilla en la que se detiene la máquina será la primera del primer grupo (es decir la del primer •)

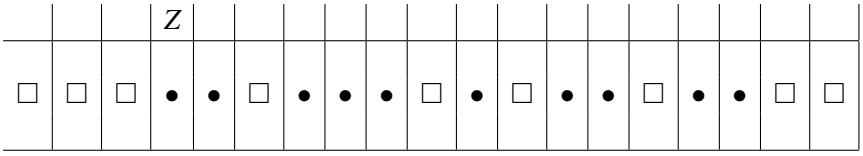
Ejemplo 5.6.1. *Cinta inicial*



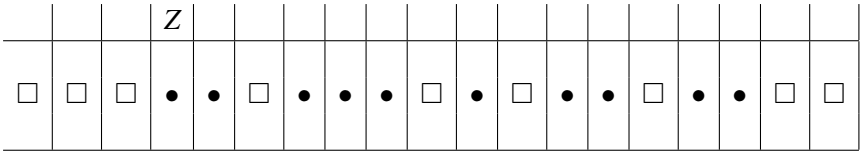
Cinta final



Ejemplo 5.6.2. *Cinta inicial*



Cinta final



Se deja como ejercicio la construcción de esta máquina. Se recomienda fuertemente tratar de construir su diagrama. Un camino posible es emplear una adaptación de la máquina que duplica vista en el ejemplo 5.2.2 para duplicar el primer y ultimo grupo. Y luego ir borrando uno de cada uno de los grupos para verificar si tienen o no la misma cantidad.

5.6.2 Representación de las funciones recursivas de lista

Mostraremos que dada una función recursiva de lista (**FRL**) existe una **MT**, con el alfabeto $A = \{\square, \bullet\}$ que tiene el mismo poder de cálculo.

Para ello debemos primero encontrar una manera de representar las listas de números en la cinta de la máquina.

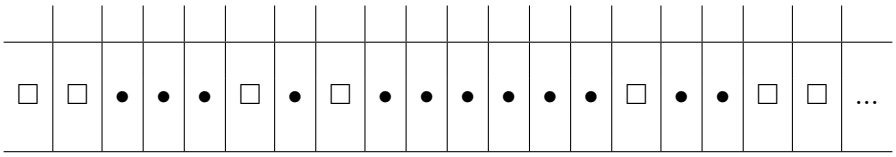
No nos queda mas camino que emplear un sistema unario, asociando al número n la cantidad de $n+1$ casillas con el símbolo $\bullet$. (eso nos permite representar el valor 0)

Para representar una lista de numeros usaremos grupos de $\bullet$ separados por un blanco.

Por ejemplo la lista

$$[2,0,5,1]$$

se escribirá en la cinta de la máquina del siguiente modo:



Notaremos la representación de una lista X de elementos pertenecientes a $\mathbb{N}_0$ en la cinta de una **MT** con $<< X >>$.

Diremos que dada una función de lista F , una máquina de Turing, que notaremos M_F, la **representa** si dada cualquier lista X que pertenezca al dominio de F, la M_F tomando como configuración inicial $<< X >>$, observando la primera casilla vacía antes del primer elemento de la lista(si es la lista vacia, en cualquier lugar) en su estado inicial, luego de procesarla llega a la configuración final

$$<< F(X) >>$$

, observando la primera casilla libre anterior a dicha configuracón.

Teorema 5.6.1. *Para toda $F \in$ **FRL**, existe siempre una $M_F \in$ **MT** que la representa.*

*Para demostrarlo usaremos un camino similar al que utilizamos para mostrar que toda **FR** era representada por una función recursiva de lista apelando esta vez a la construcción recursiva del conjunto **FRL***

I) *Las funciones base 0_i, 0_d, $\square_i$, $\square_d$,s_i,s_d son representadas por máquinas de Turing.*

En efecto veamos los diagramas de las máquinas que representan a cada una de ellas:

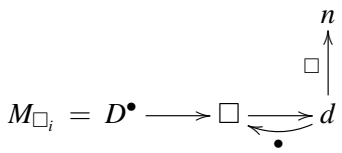
a) Cero a izquierda

$$M_{0_i} = i \ \bullet \ i$$

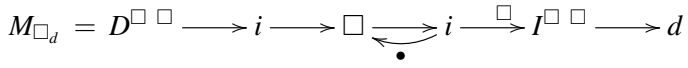
b) Cero a derecha

$$M_{0_d} = D^{\square \ \square} \ \bullet \ I^{\square\square} \ d$$

c) Blanco a izquierda



d) Blanco a derecha



e) Sucesor a izquierda

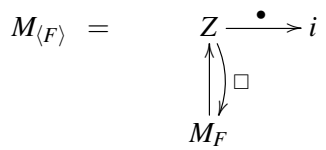
$$M_{s_i} = D^{\bullet}\ i \bullet\ i$$

f) Sucesor a derecha

$$M_{s_d} = D^{\square\ \square}\ I^{\bullet}\ d\ \bullet I^{\square\ \square}\ d$$

II) *Composición*
Dadas dos funciones de lista F y G que son representadas por ***MT*** *, M_F y M_G es inmediato que existe una* ***MT*** *(M_{FG}) que representa a la composición GF.*
En efecto M_{FG} es directamente la composición $M_F M_G$

III) *Repetición*
Finalmente, dada una función de lista F que es representada por una ***MT*** (M_F) *existe una* ***MT*** *que representa a la función $\langle F \rangle$*
Se demuestra empleando la máquina Z que vimos en 5.6.1



Esta máquina aplica M_F mientras los extremos de la lista son distintos.

Luego, hemos mostrado que cualquiera sea la ***FRL*** *existe una* ***MT*** *que la representa.*

Esto implica que como modelo de cálculo las **MT** son, al menos, tan poderosas como las **FRL**.

Turing afirmaba que cualquiera sea el cálculo, existe una **MT** que lo puede realizar. Esto es conocido como la *Tesis de Turing*.

Como habíamos visto que las **FRL** eran, al menos, tan poderosas como las **FR** aparentemente hemos llegado a un modelo que supera a las **FR**

Si utilizamos el símbolo $\preccurlyeq$ para establecer una jerarquía respecto a la capacidad de modelizar el cálculo entre los modelos que hemos visto, hasta ahora podemos poner :

$$\textbf{FR} \preccurlyeq \textbf{FRL} \preccurlyeq \textbf{MT}$$

Pero, como mostraremos a continuación, existe un modo de representar cualquier **MT** con una función de **FR** .

5.7 Representación de MT por FR

Dada una M que pertenece a **MT** diremos que una función numérica , F_M, la representa, si codificando de algun modo en forma numérica las configuraciones de la cinta, la F_M asocia al número de cada configuración inicial que puede procesar M, el número de la configuración final (es decir realiza el mismo cálculo que la máquina)

Teorema 5.7.1. *Para toda **MT** existe siempre una **función recursiva,** que la representa.*

Demostraremos, en principio, el teorema para las **MT** que tienen un alfabeto de diéz símbolos $(s_0, s_1, s_2, \ldots, s_9)$ y $p+1$ estados.

En cada momento la situación de una dada **MT** está completamente definida por los datos que contiene la cinta, la casilla observada y el estado q_e en que se encuentra

Veamos la forma de representar estos datos con un número.

Consideramos la cinta dividida en tres partes: la celda observada, y las zonas que se encuentran a izquierda y derecha de esta celda.

Supongamos una configuación como la siguiente:

								Ob										
$\ldots s_0$	s_0	s_0	s_5	s_3	s_8	s_4	s_2	s_3	s_1	s_7	s_9	s_2	s_0	s_0	s_0	s_0	s_0	$s_0 \ldots$

Podemos asociar a cada símbolo del alfabeto su subíndice (en este caso los dígitos del 0 al 9).

En nuestro ejemplo :

								Ob										
$\ldots 0$	0	0	5	3	8	4	2	3	1	7	9	2	0	0	0	0	0	$0 \ldots$

Asociamos a la parte izquierda de la cinta el número que tiene como cifras decimales los índices de los símbolos contenidos en las casillas a la izquierda de la casilla observada, teniendo en cuenta que no consideramos los ceros a la izquierda. Llamamos a este número C_i

En nuestro ejemplo será $C_i = 53842$

A la casilla observada le asociamos como número, el índice del símbolo que contiene. Notamos ese número Ob.

En nuestro ejemplo será $Ob = 3$

Finalmente asociamos a la parte derecha de la cinta el número que forman los subíndices de los símbolos que contienen **pero leyendo de derecha a izquierda**. Con lo que los que son los infinitos "ceros a la derecha" los que no cuentan en este caso. Notaremos ese número como C_d

En nuestro ejemplo será $C_d = 2971$

Por lo tanto los valores: C_i, Ob, C_d y el número del estado (un valor entre 1 y p que notaremos: e) describen la situación de la cinta en cada momento.

Asociemos a estos cuatro valores un número de Gödel (visto en 3.3.5) que llamaremos N

$$N = \overline{< C_i, Ob, C_d, e >}$$

Aplicando una función G como la vista en 3.3.4 se puede reconstruir:

$$C_i = G(0,N) \quad Ob = G(1,N) \quad C_d = G(2,N) \quad e = G(3,N)$$

A partir del estado en que se encuentra y del valor observado la máquina realiza lo que le indica su tabla y alcanza una nueva configuración y estado, $C_i^{'}, Ob^{'}, C_d^{'}, e^{'}$, al que podemos asociar un nuevo número de Gödel que llamaremos $N^{'}$.

Queremos ver cual es la naturaleza de la función, que va de N a $N^{'}$

A partir del valor observado y el estado en la tabla se obtienen tres datos: el símbolo que sutituye al observado (sea S_k - le asociamos el valor k-), el nuevo estado (sea q_f - le asociamos el valor f) y el movimiento a realizar (d, i, n).

Asociamos el valor 1 al movimiento hacia derecha, el valor 2 al movimiento a la izquierda y el valor 3 al no-movimiento. Llamamos m al valor del movimiento.

Los valores k, f y m están determinados univocamente por la tabla de la máquina.

Podemos pensar esta Tabla como tres funciones $S^{(2)}$, $E^{(2)}$ y $M^{(2)}$ definidas para los valores 0..,9 de los símbolos y para los valores $1...p$ de los estados.

Estas funciones las podemos extender para todo $\mathbb{N}_0^2$, del modo siguiente:

$$S(i,j) = E(i,j) = M(i,j) = 0 \text{ para todo } i > 0 \quad \text{o para todo } j > p$$

(p es el número de estados).

Se observa que las funciones $S^{(2)}$, $E^{(2)}$ y $M^{(2)}$ son **FRP** ya que son distintas de cero solamente en un conjunto finito de su dominio.

Veamos ahora en que modo se modifican los valores que describen la situación de la **MT** cuando se realiza un **movimiento hacia la derecha** y se sustituye el símbolo observado con $S(Ob, e)$.

Se puede ver que los nuevos valores que asociamos a las partes izquierda y derecha la cinta, al valor observado y al nuevo estado son :

$$C_i^{'} = 10.C_i + S(Ob, e)$$
$$C_d^{'} = Q(C_d, 10)$$
$$Ob^{'} = Mod(C_d, 10)$$
$$e^{'} = E(Ob, e)$$

Donde S y E son las funciones de la tabla y Q y Mod son las funciones definidas en los ejemplos 3.1.12 y 3.1.16

Si, en el ejemplo visto, el valor que sustituye al 3 es 7 y el movimiento es hacia la derecha:

									Ob									
s_0	s_0	s_0	s_5	s_3	s_8	s_4	s_2	s_7	s_1	s_7	s_9	s_2	s_0	s_0	s_0	s_0	s_0	s_0

$$C_i^{'} = 53842x10+7 = 538427$$
$$C_d^{'} = Q(2971,10) = 297$$
$$Ob^{'} = Mod(2971,10) = 1$$
$$e^{'} = E(Ob,e)$$

Si el movimiento es **hacia la izquierda:**

$$C_i^{'} = Q(C_i,10)$$
$$C_d^{'} = C_d x10+S(Ob,e)$$
$$Ob^{'} = Mod(C_i,10)$$
$$e^{'} = E(Ob,e)$$

y si no se mueve:

$$C_i^{'} = C_i$$
$$C_d^{'} = C_d$$
$$Ob^{'} = S(Ob,e)$$
$$e^{'} = E(Ob,e)$$

Veamos ahora como calculamos el valor de $N^{'}$ en cada caso:

Si el movimiento es hacia la derecha(llamaremos al número $N^{'}{}_d$)

$$N^{'}{}_d = \overline{< C_i^{'}, Ob^{'}, C_d^{'}, e^{'} >} = \overline{< 10.C_i+S(Ob,e), Mod(C_d,10), Q(C_d,10), E(Ob,e) >}$$

$$N^{'}{}_d = \overline{< 10.G(0,N)+S(G(1,N),G(3,N)), Mod(G(2,N),10), Q(G(2,N),10), E(G(1,N),G(3,N)) >}$$

Si el movimiento es hacia la izquierda (llamaremos al número $N^{'}{}_i$)

$$N^{'}{}_i = \overline{< Q(G(0,N),10), Mod(G(0,N),10), 10.G(2,N)+S(G(1,N),G(3,N)), E(G(1,N),G(3,N)) >}$$

Si no se mueve (llamaremos al número $N^{'}{}_n$)

$$N^{'}{}_n = \overline{< G(0,N), S(G(1,N),G(3,N)), G(2,N), E(G(1,N),G(3,N)) >}$$

Para contemplar las tres posibilidades basta usar el valor del movimiento para anular los casos que no nos interesan:

$$N^{'} = N^{'}{}_{d}((m-2)(m-3)/2) + N^{'}{}_{d}((m-1)(m-3)/(-1)) + N^{'}{}_{n}((m-1)(m-2)/2)$$

Vimos que $N^{'}{}_{d}$, $N^{'}{}_{i}$ y $N^{'}{}_{n}$ son valores que dependen sólo del valor de N y de la tabla de la **MT**.

Como $m = M(Ob,e) = M(G(1,N),G(3,N))$, el valor $N^{'}$ se puede obtener como resultado de aplicar una función que llamaremos F_M al valor N (ademas esta función es **FRP**)

Podemos, a partir de una configuración inicial de la cinta que llamaremos C_1, obtener el número de Gödel que corresponde al estado inicial. Lo llamaremos N_1

$$N_1 = \overline{< C_{i1}, Ob_1, C_{d1}, 1 >}$$

Al aplicar la función F_M repetidas veces en el paso t el valor que indica la situación de la **MT** será:

$$N_t = F_M{}^t(N_1)$$

Si en algún momento llegamos al estado q_0 hemos completado el cálculo.

El número de pasos necesarios para llegar a la configuración final (si eso es posible) lo llamamos t_f.

El valor de t_f (si existe) es el mínimo valor de t tal que $G(F_M{}^t(N_1),3) = 0$

O sea que encontramos t_f aplicando el minimizador a la $G(F_M{}^t(N_1),3)$

Se trata de una **FR** .

Para ese valor de t_f encontramos la configuración final de la cinta, C_f

$$C_{f_i} = G(F_M{}^{t_f}(N_1),0)$$

$$C_{f_d} = G(F_M{}^{t_f}(N_1),2)$$

$$Ob_{f_i} = G(F_M{}^{t_f}(N_1),1)$$

Luego hemos mostrado que dada una **MT** existe una **FR** capáz de realizar el mismo cálculo.

(La condición de que el alfabeto tenga diez valores no es ninguna limitación. En el caso de un alfabeto de k elementos basta tomar el sitema numérico de base k . El valor 10 de las fórmulas sigue valiendo pues en cada caso es la forma de expresar el valor de la base)

5.8 Tesis de Church-Turing

Con lo demostrado en el teorema anterior la jerarquía respecto a la capacidad de modelizar el cálculo entre los modelos vistos queda :

$$\mathbf{FR} \preccurlyeq \mathbf{FRL} \preccurlyeq \mathbf{MT} \preccurlyeq \mathbf{FR}$$

Con lo que se muestra que todos los modelos vistos son equivalentes.

En el desarrollo de otros modelos (por ejemplo : lenguajes de programación, lógica combinatoria, λ cálculo, máquinas de Turing con varias cintas) no se ha conseguido superar los vistos, por lo que se fortalece la tesis, que ahora llamaremos de **Church-Turing** de que todo proceso algorítmico puede ser representado en los modelos vistos.

Que el convencimiento sea generalizado entre lógicos y matemáticos no significa que se tenga la certeza ya que estos conceptos no se definen por sufragio universal. (se invita al lector a seguir buscando alternativas).

5.9 Los límites del cálculo

Veremos que existen funciones naturales que no son calculables.

Nos basta con mostrar que hay funciones naturales que no son **FR**

Observemos que por definición, cada **FR** está identificada con una fórmula, la cual combina funciones básicas y operadores.

El conjunto de símbolos que representan las funciones base y los operadores es numerable.

Las combinaciones finitas de estos símbolos tomados de un conjunto numerable forman un conjunto numerable.

No cualquier combinación finita de símbolos representa a una **FR**.

Por lo tanto, la cardinalidad de las **FR** es menor o igual que la de los conjuntos finitos de símbolos. Dado que las **FR** son no-finitas, la cardinalidad de las **FR** es $\aleph_0$.

Calculemos ahora la cardinalidad de las funciones naturales. Es fácil ver que las funciones cuyo conjunto recorrido es el $\{0,1\}$ conforman un subconjunto propio de las funciones naturales. Estas funciones son las conocidas como funciones características, las cuales tienen una correspondencia biunívoca con el conjunto de partes de $\mathbb{N}$.

Pero la cardinalidad de partes de $\mathbb{N}$ es $\aleph_1$. Luego, la cardinalidad de las funciones características es $\aleph_1$.

Al ser el conjunto de las funciones características un subconjunto propio de las funciones naturales, podemos concluir que estas últimas tienen, al menos, cardinalidad $\aleph_1$.

Por lo tanto, existen funciones naturales que no son funciones recursivas.

Aunque hemos utilizdo las **FR** esta demostración es independiente de la Tesis Church-Turing. Aun cuando se descubra que esa tesis no es válida, el concepto de algorítmo como conjunto finito de instrucciones sigue teniendo la misma limitación. Hoy que estamos acostumbrados a la digitalización podemos pensar cualquier conjunto de instrucciones, escrito o contado en cualquier idioma, o mostrado en una completa y detallada filmación, soportada en un enorme número binario. Dado que conjunto de esos números tiene cardinalidad numerable, podemos mantener que existen funciones naturales que no son calculables independientemente del modelo.

5.10 El problema de la parada

Sabemos que hay funciones que no son calculables, mas aun, mostramos que hay un orden de infinitud que las separa de las calculables, pero no hemos mostrado ninguna.

Pero resultaría muy interesante encontrar una función, de la que se pueda describir que es lo que hace, y mostrar que no podemos calcular.

Turing, nos presenta una función de ese tipo en el llamado: “problema de la parada” (halting problem).

Notemos que, dependiendo de cada **MT**, no siempre ante una configuración incial se llega a una configuración final.

Por ejemplo la máquina $D^{\bullet\bullet}$, con una cinta inicial que no tenga dos $\bullet$ seguidos, no se detendrá nunca.

La base de la idea es si se puede **calcular**, para cualquier **MT** y configuración inicial dadas, si la máquina llega a un resultado final o no se detiene nunca. Observemos que no nos preguntamos sobre el resultado, sino si se detiene o no. Para casos particulares podemos saber lo que sucede sin necesidad de realizar el cálculo.

La pregunta es si existe un método para contestar esta pregunta ante cualquier caso.

(Observemos que no sirve como técnica el poner a trabajar la máquina, pues, no sabemos si terminará o no)

Acotaremos el problema limitandonos a todas las posibles **MT** que reciban como dato de entrada un número.

A cada una de esas **MT** se le puede asociar un número natural que podemos calcular.

Las maneras como podemos hacer esto son diversas, pero apelaremos a una consideración muy sencilla:

Imaginemos que escribimos la tabla de la máquina con un procesador de texto cualquiera. El archivo donde guardamos esa información será una larga fila de ceros y unos. Podemos asociar ahora a cada máquina el número ternario que se forma poniendo el número dos delante de dicha lista de dígitos.

Dada una mq́uina de Turing M, llamaremos : N_M al número que asociamos a dicha máquina.

Turing planteó la siguiente función:

Dado un número natural cualquiera nos fijamos si representa una **MT** de acuerdo a nuestro método (la posibilidad, bajísima, que ese número, presentado en forma ternaria sea un 2 seguido de una lista de ceros y unos, y, que además, esa lista sea la de un archivo que corresponda a la tabla de una **MT**.).

Si esto no sucede asociamos a este número el valor cero.

Si nos encontramos ante una **MT**, digamos M_k , podemos pensar en procesar con esta máquina su propio número.

Asociamos un 1 al número si la máquina M_k termina el cálculo tomando como dato inicial su número N_{M_k}

En caso de que la máquina no termine, asociamos el 2 al número inicial.

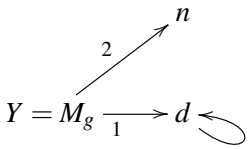
Hemos definido una función g que va de los naturales a los naturales.

Turing muestra que esta función no es calculable de la siguiente manera:

Si g fuese calculable existiría una máquina de Turing, M_g, que realizaría el cálculo de dicha función.

Esto es, ante cualquier máquina de Turing M, dado el número asociado N_M, nuestra máquina tomaría como dato inicial dicho número N_M y terminaría su trabajo mostrando un 1 si la máquina M termina cuando procesa el valor N_M como valor inicial, o un 2 en caso de que esto no suceda.

Si esa máquina existe podemos construir, sencillamente, una máquina que llamaremos Y de la siguiente manera:



Pero esta máquina Y tendrá asociado un número N_Y.

La pregunta es que sucede si esta máquina Y procesa su propio número .

Si aplicamos al número N_Y la, que suponemos existe, máquina, M_g nos dará como resultado un 1 si la máquina Y termina con un resultado al procesar N_Y.

Pero, si aplicamos directamente la máquina Y a su número N_Y, como composición de dos máquinas primero , procesa M_g y termina con un 1 , esta rama va a un ciclo infinito de moverse hacia la derecha... no termina.

Si el resultado para la M_g es que Y no se detiene al procesar el N_Y , observemos que en este caso la máquina termina...

Este resultado contradictorio, muestra que no es posible construir una máquina que pueda calcular la función g.

Hemos encontrado una función no calculable.

5.11 Ejercicios

Ejercicio 5.11.1.

Defina tabla de transiciones y diagramas para las siguientes máquinas de Turing:

(a) $I^{\square\square}$, que se mueve a la izquierda hasta encontrar dos símbolos $\square$ consecutivos

(b) $D^{\bullet\bullet}$, que se mueve a la derecha hasta encontrar dos símbolos $\bullet$ consecutivos

Ejercicio 5.11.2. Defina una máquina de Turing sobre el alfabeto $\{\square, \bullet, \circ\}$ que se comporte de acuerdo a la siguiente especificación:

- Si $x = y$ se detenga si y sólo si la cadena de entrada contiene un número par de símbolos $\circ$
- Si $x \neq y$ se detenga si y sólo si la entrada es de la forma $\bullet^n \circ^n$, para algún $n \in \mathbb{N}$

donde x e y son el primer y el último símbolo de la cinta de entrada, respectivamente.

Asuma que la entrada está formada por una sola cadena de $\{\bullet, \circ\}^*$

Ejercicio 5.11.3. Defina una máquina de Turing que calcule el resto de dividir por 3 una secuencia de $\bullet$. La máquina inicia con una secuencia de $\bullet$, y deberá terminar con una secuencia de $\bullet$ que tendrá 0, 1, o 2 elementos según si la cantidad original de $\bullet$ tenia resto 0, 1 o 2 al dividir por 3. La máquina iniciará sobre el $\square$ inmediato a la izquierda de la secuencia. Por ejemplo, al ejecutar sobre:

$$\ldots\square\underline{\square}\bullet\bullet\bullet\bullet\bullet\square\square\ldots$$

obtendremos como resultado:

$$\ldots\square\underline{\square}\bullet\bullet\square\square\ldots$$

Ejercicio 5.11.4. Construya una máquina de Turing que dado un número natural $n > 0$, escrito en binario, escriba en la cinta $n+2$, también representado en binario. Asuma que el alfabeto es $\{0, 1, \square\}$, que el dígito menos significativo está a la derecha, y el más significativo a la izquierda. La máquina comienza apuntando al dígito más significativo, y deberá terminar en el dígito más significativo del número resultante.
Por ejemplo, la configuración incial de la cinta para el número $11 = 8+2+1$ será:

$$\ldots\square\square\square\underline{1}011\square\square\square\ldots$$

y luego de ejecutar la máquina deberá ser $13 = 8+4+1$, es decir, en la cinta:

$$\ldots\square\square\square\underline{1}101\square\square\square\ldots$$

Nota: si $n = b_k 2^k + b_{k-1}2^{k-1} + \ldots + b_1 2^1 + b_0 2^0$ con $k = \lfloor log_2(n) \rfloor$ y cada $b_i \in \{0,1\}$ entonces su representación en binario es $b_k b_{k-1} \ldots b_1 b_0$.

Ejercicio 5.11.5. Estamos interesados en definir una máquina de Turing que decida si una palabra sobre el alfabeto $\{a, b\}$ que recibe como entrada pertenece a determinado lenguaje $\mathscr{L}$ sobre dicho alfabeto. Para esto, utilizaremos como símbolos de cinta $\{\square, a, b, \circ, \bullet\}$. La cinta de entrada de la máquina contendrá una palabra $\alpha \in \{a, b\}^\star$, y la configuración final será de la forma:

$$\ldots\square\square\circ\square\square\ldots$$

en caso de que la palabra pertenezca al lenguaje, y con la cinta en forma

$$\ldots\square\square\bullet\square\square\ldots$$

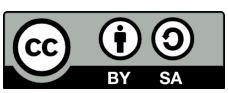
en caso de que no pertenezca. En caso que para un lenguaje $\mathscr{L}$ exista una máquina MT con estas características, decimos que $\mathscr{L}$ es decidible.

Muestre que los siguientes lenguajes son decidibles:

1. $\mathscr{L}_0 = \{aa, b\}$
2. $\mathscr{L}_1 = \{a^n b a^m \mid n, m \in \mathbb{N}_0\}$
3. $\mathscr{L}_2 = \{a^n b^m \mid n, m \in \mathbb{N}_0\}$
4. $\mathscr{L}_3 = \{a^n b^n \mid n \in \mathbb{N}_0\}$
5. $\mathscr{L}_4 = \mathscr{L}_1 \cup \mathscr{L}_2$

Ejercicio 5.11.6. Construya la máquina **Z**

Edición: Marzo de 2014.
Este texto forma parte de la Iniciativa Latinoamericana de Libros de Texto abiertos (LATIn), proyecto financiado por la Unión Europea en el marco de su Programa ALFA III EuropeAid.



Los textos de este libro se distribuyen bajo una Licencia Reconocimiento-CompartirIgual 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/deed.es_ES