BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS
LICENCIATURA EN MATEMÁTICAS

ESTRUCTURAS DESDE EL PUNTO DE VISTA DE LA TEORÍA DE LA COMPUTABILIDAD

TESIS

QUE PARA OBTENER EL GRADO DE
LICENCIADO EN MATEMÁTICAS

PRESENTA
LUIS FERNANDO ALTAMIRANO FERNÁNDEZ

DIRECTOR DE TESIS
DR. IVÁN MARTÍNEZ RUÍZ

H.PUEBLA DE ZARAGOZA, PUE. FEBRERO 2021

Agradecimientos

Quiero agradecer a mi abuelita Toña y a mi mamá por todo su amor y por todo su apoyo. Quiero agradecerles también por todas sus grandes enseñanzas y por ser mi inspiración para ser mejor cada día. No puedo ser más afortunado por el hecho de que Dios las haya puesto en mi camino. Siempre las voy a llevar en mi corazón.

A mi papá, a mis abuelos y a toda mi familia en Tuxtepec por siempre creer en mí y por estar ahí siempre que lo he necesitado. En particular gracias a mi papá por todo su apoyo y por todos sus consejos que siempre me han ayudado a salir adelante.

A mis hermanas, a mi abuelo y a toda mi familia en Puebla por enseñarme el trabajo duro, el pensamiento crítico y a siempre ser solidario y empático con los demás. Quiero hacer un agradecimiento especial a mi tío Jorge por prestarme sus libros de matemáticas y así inspirarme el gusto por esta bella disciplina y por todo el apoyo prestado durante la licenciatura.

A mis amigos: a las lobas, a la secta y a los de la facultad por todas las vivencias y aprendizajes que hemos tenido juntos.

A mi asesor el Dr. Iván Martínez Ruíz por haber aceptado que fuera su tesista, por su tiempo y sobre todo por su paciencia durante el desarrollo y revisión de cada uno de los detalles de los resultados que aquí se presentan.

A mis sinodales el Dr. César Bautista Ramos, el Dr. Antonio Montalbán y el Dr. Alejandro Ramírez Páramo por su tiempo y paciencia para la lectura y revisión de este trabajo.

Resumen

La teoría de la computabilidad se centra en el estudio de la complejidad de los objetos matemáticos en función de la dificultad de dar un algoritmo para calcularlos. Después del trabajo fundacional en el área de Turing en 1936, hubo desarrollos de Kleene, Post, Turing, Church y Markov que sirvieron de base a la teoría. Algunos de estos resultados se pueden encontrar en [Dav]. En este trabajo tomamos algunas de las definiciones básicas de allí y las modificamos para darle al lector una mirada más completa. Otro enfoque que usa un tipo diferente de máquina se puede encontrar en [Cut]. En este trabajo modificamos una máquina de Turing añadiéndole direccionamiento indirecto y así convirtiéndola en una máquina de acceso aleatorio. También presentamos de manera formal las principales definiciones y resultados dados en [Cut]. Nuestros propósitos principales son: el mostrar la existencia de un programa universal, mostrar ejemplos explícitos de conjuntos no computables, introducir la noción de que un conjunto sea enumerado por una función computable e introducir los grados de Turing.

Como aplicación de la teoría, se plantean interrogantes sobre las propiedades de computabilidad de las estructuras matemáticas como el problema de la palabra de grupos, la existencia de campos isomorfos de forma no computable y en general la existencia de copias isomorfas no computables de una estructura desde un punto de vista sintáctico. En este trabajo definimos cuando una estructura es computable en relación con un oráculo y mostramos algunas propiedades. Después de esto, abordamos el problema de que dos estructuras isomorfas no necesariamente tienen las mismas propiedades de computabilidad, tales como tener el mismo grado de Turing y tener relaciones que son computables a partir de una pero no a partir de la otra. Exploramos una razón por la que esto podría suceder y es que entre estructuras isomorfas no hay necesariamente un isomorfismo computable. Observamos este problema cuando una estructura es computable. Estudiamos una caracterización sintáctica de una solución uniforme, en el sentido de obtener los isomorfismos de un programa único que puede utilizar como oráculo el grado de Turing de la otra estructura isomorfa y también estudiamos una caracterización sintáctica de la respectiva solución no uniforme. En este trabajo seguimos la línea de estudio de [Mon] y ampliamos algunas pruebas mostradas ahí.

Introducción

¿Qué es un *algoritmo*? En este trabajo adoptamos la definición que da la RAE: «es un conjunto ordenado y finito de operaciones que permite hallar la solución de un problema».

La noción de algoritmo ha acompañado a la humanidad desde sus primeros años de existencia. Después de suficiente experiencia, empezamos a seguir una secuencia de pasos determinados para poder manufacturar las primeras herramientas de piedra, también para poder controlar el fuego, para construir armas sofisticadas como el arco y la flecha y para llevar a cabo labores más sofisticadas como la agricultura.

Si nos adelantamos más en el tiempo también tenemos como ejemplos a la serie de pasos que seguían los egipcios para obtener el volumen de pirámides truncadas y a la serie de pasos, conocida como criba de Eratóstenes, que se debía seguir para encontrar los números primos menores que un número dado.

La idea de realizar una serie de pasos determinados y en ocasiones repetitivos para resolver una tarea nos trajo la idea de construir máquinas que de forma automática pudieran realizar estas tareas. Como ejemplo tenemos a las máquinas calculadoras construidas a mediados del siglo XVII por el matemático Blaise Pascal y de manera independiente por el matemático y filósofo Gottfried Leibniz.

La idea de obtener resultados a partir de una serie mecanizada de cálculos no sólo se limitó a obtener resultados de operaciones aritméticas sino también a obtener deducciones de un conjunto de premisas. Leibniz, también, dentro de sus cavilaciones se preguntaba si era posible construir un lenguaje universal en el que se pudiera expresar todo el conocimiento humano de una forma sencilla y simple y que además se pudieran derivar verdades dentro de este lenguaje a partir de una serie de cálculos.

En 1928, David Hilbert en una postulación de un problema similar a esta cuestión, se planteó si dado un sistema lógico, es posible encontrar un método efectivo o algoritmo general para que dado un enunciado en el lenguaje del sistema éste determine si el enunciado es demostrable en el sistema o no.

Para dar una respuesta a este problema era necesario dar una definición formal

en términos del lenguaje matemático de la noción informal de método efectivo o de algoritmo. En 1936 el matemático Alonzo Church en su artículo titulado *λ-definibilidad y recursividad* describió un sistema formal denominado λ-cálculo sobre el cual afirmó que la noción de una función calculable efectivamente era aquella que era λ-definible en su sistema. Dentro de este sistema pudo dar una respuesta negativa al problema de Hilbert.

Por otra parte, en el mismo año el matemático Alan Turing en su artículo de nombre *On computable numbers an its applications to the Entscheidungsproblem* define un objeto matemático denominado hoy en día como *máquina de Turing* sobre la que afirmó que captura la noción informal de algoritmo. En este artículo, al igual que Church, también da una respuesta negativa al problema de Hilbert. Más tarde en 1937 en su artículo *Computabilidad y λ-definibilidad*, Alan Turing demostró que el λ-cálculo de Church era equivalente a su sistema. Por estas razones, a la creencia que estas nociones describen formalmente la noción informal de algoritmo se le conoce como *Tesis de Church-Turing*.

En los años siguientes, lógicos y matemáticos como Stephen Kleene, Emil Post, Alan Turing, Alonzo Church y Andrey Markov en una serie de artículos y libros cimentaron las bases de lo que hoy en día conocemos como Teoría de la Computabilidad.

Con los avances obtenidos hasta ese momento en esta área y en álgebra, los lógicos y matemáticos Piotr Nóvikov y William Boone resolvieron respectivamente de manera independiente en sus artículos *On the algorithmic unsolvability of the word problem in group theory* y *The word problem* de manera negativa el *problema de la palabra*.

Siguiendo en esta línea, el matemático Serguey Ivanovich Adyan en su artículo *Algorithmic unsolvability of problems of recognition of certain properties of groups* continúo con el estudio de la existencia de algoritmos para resolver distintos problemas en teoría de grupos.

En 1956, de manera conjunta los matemáticos Albrecht Fröhlich y John Shepherdson en su artículo *Effective procedures in field theory* dieron respuesta a problemas de aplicación de la teoría de la computabilidad en la teoría de campos, incluyendo la demostración de la existencia de estructuras isomorfas pero que no son isomorfas de manera computable.

A principios de los años 60, por una parte el matemático Michael Rabin en su artículo *Computable algebra, general theory and theory of computable fields* y por otra parte el matemático Anatoli Máltsev en sus artículos *Constructive algebras I* y *On recursive abelian groups* estudiaron problemas de decidibilidad en otras

estructuras algebraicas.

En los años 70, los matemáticos George Metakides y Anil Nerode e independientemente los matemáticos de la escuela siberiana de matemáticas constructivas como Yuri Ershov utilizaron nuevas técnicas para ampliar el estudio de más tipos de estructuras en términos de la teoría de la computabilidad.

En años posteriores se continúo con el estudio de técnicas para resolver y analizar problemas en esta rama de la teoría de estructuras utilizando fórmulas de longitud infinita para estudiar la complejidad de éstas. A mediados de los años 90 se realizó un compendio realizado en conjunto por los matemáticos Chris Ash y Julia Knight titulado *Computable Structures and the Hyperarithmetical Hierarchy* en el cual se muestra el trabajo desarrollado que de entre los demás problemas que aborda, el problema de encontrar condiciones sintácticas para que exista un isomorfismo computable entre dos estructuras computables isomorfas.

Recientemente se ha publicado un libro hecho por el matemático Antonio Montalbán titulado *Computable Structure Theory: Within arithmetic* que aborda la problemática antes mencionada y que tiene puntos en común con el libro de Chris Ash y Julia Knight pero contiene otra forma de presentar los resultados.

En el capítulo I mostraremos las bases de la Teoría de la Computabilidad enunciando primero el modelo de cómputo de máquinas de Turing. Este modelo es el que con mayor facilidad puede ser explicado formalmente y similar al que Alan Turing propuso en 1936. Definiremos nociones de computabilidad para funciones usando este modelo. Posteriormente daremos un modelo de cómputo con mayor facilidad de ser utilizado para demostrar resultados más importantes dentro de la teoría. También daremos una noción de computabilidad para funciones con este modelo. Y hablaremos de la relación que existe con la noción dada por el otro modelo. Mostraremos propiedades de funciones computables y veremos que podemos asignar de forma computable un número natural en específico a un programa. El estudio de máquinas de Turing está basado en [Dav]. Los demás temas que aparecen en este capítulo están basados en [Cut] y además son presentados con un enfoque distinto.

En el capítulo II daremos resultados importantes como el de mostrar la existencia de un programa que puede obtener el resultado de cualquier otro en cualquier entrada siempre y cuando estos últimos esten definidos. También mostraremos ejemplos explícitos de conjuntos que no son computables y una herramienta que nos facilitará dar ejemplos explícitos de este tipo de conjuntos. Los temas presentados en este capítulo están basados en [Cut].

En el capítulo III daremos una definición formal de la noción de enumerar un conjunto de forma computable, también daremos la definición de un par de

relaciones de equivalencia sobre los conjuntos de números naturales denominadas m equivalencia y Turing equivalencia respectivamente. El tema de computabilidad enumerable está basado en [Dav] y en [Cut].

En el capítulo IV damos los elementos necesarios de la teoría de estructuras y modelos. Este material está basado en [CK]. Posteriormente damos una forma de codificar fórmulas de un lenguaje de primer orden con números naturales, de modo que podamos dar una definición de computabilidad de una estructura. Daremos algunas propiedades y mostraremos que en términos de computabilidad podemos suponer que una estructura sólo tiene relaciones. La presentación de esta parte está basada en [Mon] y aborda detalles sobre algunas definiciones y demostraciones.

En el capítulo V abordamos el problema de la existencia de estructuras que son isomorfas pero que tienen distintas propiedades de computabilidad como lo es el tener distinto grado de Turing. Abordaremos una razón por la que esto puede ocurrir, como lo es el que no exista un isomorfismo computable entre dos estructuras. Daremos dos abordajes a esta problemática las cuales denominaremos como versión uniforme y versión no uniforme. En la versión uniforme mostraremos una caracterización sintáctica de la existencia de un solo programa que nos produzca isomorfismos entre una estructura computable y sus estructuras isomorfas contables utilizando como oráculo a éstas últimas. En la versión no uniforme daremos una caracterización sintáctica similar a la versión uniforme, sólo que evitaremos el hecho de que los isomorfismos se puedan producir con un solo programa. Para ello añadiremos un nuevo tipo de lenguaje para estudiar nuestras estructuras. Este capítulo se basa en [Mon] y se amplían algunas definiciones y demostraciones.

Notación y Preliminares

Dados dos conjuntos A, B diremos que $A \subseteq B$ si y sólo si para todo $x \in A$ implicamos que $x \in B$. Diremos que $A = B$ si y sólo si $A \subseteq B$ y $B \subseteq A$. Denotamos por $A \cup B$ al conjunto unión de A con B; $A \cap B$ denota al conjunto intersección de A con B y $A \setminus B$ al conjunto de elementos que están en A pero que no están en B.

Dados un conjunto I y un conjunto de conjuntos $\mathcal{F} = \{A_n : n \in I\}$, denotamos a la unión de $\mathcal{F}$ por $\bigcup \mathcal{F}$, que también denotamos por $\bigcup\limits_{n \in I} A_n$. Si $I = \mathbb{N}$ entonces la unión la denotamos por $\bigcup\limits_{n=1}^{\infty} A_n$.

De manera similar denotamos la intersección de $\mathcal{F}$ como $\bigcap \mathcal{F}$ ó como $\bigcap\limits_{n \in I} A_n$, ó bien si $I = \mathbb{N}$ por $\bigcap\limits_{n=1}^{\infty} A_n$.

Dados dos conjuntos A y B denotamos por $f : A \to B$ a una función f con dominio A y codominio B. Al conjunto de funciones con dominio A y codominio B lo denotamos por ${}^{B}A$. Si el dominio de la función f es un subconjunto de A entonces denotamos a f como $f : A \nrightarrow B$ y diremos que f es una función parcial de A a B. Observemos que en particular una función es un caso particular de una función parcial. Si f, g son funciones parciales, diremos que $f \subseteq g$ si y sólo si dom $f \subseteq$ dom g y para todo x en dom f tenemos que $f(x) = g(x)$. Si $f : A \nrightarrow B$ es una función parcial, denotamos por dom f al dominio de f y por im f a la imagen de f que es el conjunto $\{y \in B \mid \exists x \in \text{dom } f \ (f(x) = y)\}$.

Denotamos por ω al conjunto de números naturales $\{0, 1, 2, 3, ...\}$. Por $\mathbb{N}$ denotamos al conjunto de números naturales menos el elemento 0, es decir al conjunto $\{1, 2, 3, ...\}$. Denotamos por $\mathbf{n}$ al conjunto de los primeros n números naturales $\{0, 1, ..., n-1\}$, es decir $\mathbf{n}$ es el número natural n visto como ordinal.

Dados un conjunto A y $n \in \omega$ con $n \geq 1$, definimos $A^n = \{(k_0, ..., k_{n-1}) : k_0, ..., k_{n-1} \in A\}$ como el conjunto de tuplas de A de tamaño n. Denotamos por $A^{<\mathbb{N}} = \bigcup\limits_{i=1}^{\infty} A^n$, es decir $A^{<\mathbb{N}}$ es el conjunto de tuplas finitas no vacías de elementos de A.

Un elemento $\gamma \in A^{<\mathbb{N}}$ de la forma $(k_0, ..., k_{n-1})$ también lo podemos ver como una función $f \in {}^{\mathbf{n}}A$ cuyo dominio es $\{0, 1, ..., n-1\}$ y es tal que $f(i) = k_i$ para cada $0 \leq i \leq n-1$. Y viceversa, una función $f \in {}^{\mathbf{n}}A$ con dominio $\{0, 1, ..., n\}$ la podemos ver como la tupla finita $(f(0), ..., f(n-1))$. Por lo tanto en ocasiones consideraremos a ambas definiciones de manera indistinta. En cualquier caso diremos también que γ es una sucesión finita de elementos de A. Si $\gamma = (k_0, ..., k_{n-1})$ denotaremos por γ_i a k_i, el cual también denominaremos i-ésimo elemento de γ. Usando la notación previa, también denotaremos a γ como $\{\gamma_0, ..., \gamma_{n-1}\}$.

En ocasiones nos referiremos al dominio de γ como el dominio de su función equivalente, el cual equivaldría a los primeros $n-1$ elementos si la longitud de γ, la cual denotamos por $len(\gamma)$, es n y a la imagen de γ como la imagen de su función equivalente.

Dados una sucesión finita γ de longitud n y $k \leq n-1$, denotamos por $\gamma \restriction k$ a la tupla $(\gamma(0), ..., \gamma(k))$. Nos referiremos a esta tupla como la tupla γ truncada en sus primeros k elementos. Si $\sigma = (k_0, ..., k_m)$ es una tupla finita de longitud m con elementos en ω tal que para cada $0 \leq i \leq m$ se tiene que $0 \leq k_i \leq n-1$, denotamos por $\gamma \restriction \sigma$ a la tupla $(\gamma(k_0), ..., \gamma(k_m))$. En ocasiones, de preferencia, pediremos que la longitud de σ sea menor o igual que la longitud de γ.

Dadas dos sucesiones finitas σ, $\gamma \in A^{<\mathbb{N}}$, diremos que $\sigma \subseteq \gamma$ si y sólo si σ está contenida en γ si tanto σ como γ son vistas como funciones.

Dada una familia de conjuntos $\{A_n : n \in I\}$ denotamos por $\prod_{n \in I} A_n$ al conjunto $\{f : I \to \bigcup_{n \in I} A_n : \forall n \in I \ (f(n) \in A_n)\}$. Si $I = \omega$ entonces al producto anterior también lo denotamos por $\prod_{n \in \omega} A_n$. Si $I = \omega$ y tenemos que cada $A_n = A$ para algún conjunto A entonces al producto lo denotamos por ${}^{\omega}A$.

Similar al caso de tuplas finitas, si tenemos un conjunto A, a cada función $f \in {}^{\omega}A$ le podemos asociar una tupla «infinita» γ de longitud ω que satisface que para cada $n \in \omega$, $f(0), ..., f(n-1)$ son los primeros n elementos de γ en el orden en el que se presentan. En este caso a γ, la podemos denotar también como $(f(0), f(1), f(2), ...)$. Y viceversa a una tupla infinita γ digamos de la forma $(k_0, k_1, k_2, ...)$ le podemos asociar la función $f \in {}^{\omega}A$ tal que $f(i) = k(i)$ para cada $i \in \omega$. Por lo tanto podemos ver a una función en $f \in {}^{\omega}A$ como a una tupla infinita y viceversa. Por consiguiente consideraremos a ambas definiciones de manera indistinta. En ambos casos diremos que tenemos una sucesión infinita de longitud ω con elementos en A. Si $\gamma = (k_0, k_1, k_2, ...)$ denotaremos por γ_i a k_i, el cual también denominaremos i-ésimo elemento de γ. También denotaremos a γ como $\{\gamma_i : i \in \omega\}$.

Dados una sucesión $\gamma \in {}^{\omega}A$ y $n \in \omega$, denotamos por $\gamma \restriction k$ a la sucesión finita $\{\gamma_0, \gamma_1, ..., \gamma_{n-1}\}$. Si σ es una sucesión finita de longitud m y todo elemento de *sigma* está en ω entonces denotamos por $\sigma \restriction \sigma$ a la tupla finita $(\gamma(\sigma(0)), ..., \gamma(\sigma(n-1)))$.

Si γ es una sucesión infinita en ${}^{\omega}A$ y σ es una sucesión finita ó infinita entonces diremos que $\sigma \subseteq \gamma$ si y sólo si σ está contenida en γ cuando ambas son vistas como funciones.

Índice general

Capítulo 1

Introducción a la Teoría de la Computabilidad

1.1. Máquinas de Turing y Máquinas de Acceso Aleatorio

En este trabajo entenderemos de manera informal que un *algoritmo* es un conjunto ordenado y finito de *instrucciones* que tras ser ejecutadas en el orden en el que se presentan o que éstas indiquen nos permitan resolver un determinado *problema*.

Con la ejecución de las instrucciones se realizan una o varias transformaciones sobre determinados *objetos*, las cuales en algunos casos para ejecutarse necesitan *información extra* sobre éstos u otros objetos.

Supondremos que los objetos en cuestión son finitos. Por lo tanto podremos asociarles una sucesión finita de símbolos o bien un número natural. A la información extra que se requiera la interpretaremos mediante conjuntos de números naturales. De este modo, en nuestro estudio nos interesaremos únicamente en numeros naturales como objetos, los cuales representaremos como una sucesión finita de simbolos tomados de un alfabeto previamente determinado o simplemente como números naturales.

Nuestra intención será representar formalmente las transformaciones mediante la manipulación de sucesiones de símbolos en base a instrucciones con una sintaxis previamente definida. A esta representación la denominaremos *modelo de cómputo*. Para ello, primero, describiremos gráficamente al modelo de cómputo, de modo que podamos tener una idea de su comportamiento y posteriormente haremos una descripción formal.

Comenzaremos con el modelo de *Máquinas de Turing*, posteriormente introduciremos el de *Máquinas de Acceso Aleatorio* que también llamamos *RAM* por sus siglas en inglés.

1.1.1. Máquinas de Turing

Gráficamente, una *máquina de Turing* es una *máquina* que consiste de una *cinta* bidimensional horizontal dividida en *celdas* del mismo tamaño y un *cabezal* sobre la cinta tal como se muestra en la figura 1.1.

Pedimos que la cinta sea potencialmente infinita en longitud en ambas direcciones, es decir que siempre que lo necesitemos siempre seamos capaces de extender la cinta añadiendo celdas a la derecha o izquierda.

Las celdas de la cinta pueden contener o bien un símbolo de un alfabeto elegido previamente o bien se puede encontrar en blanco (sin ningún símbolo).

El cabezal de la cinta se posa sobre una y sólo una celda de la cinta. Éste contiene el *estado actual* de la máquina, una cantidad finita de *instrucciones* y posibles símbolos y estados de la máquina. También puede recuperar y modificar el símbolo que se encuentra en la cinta en un momento dado y además puede saber cuántos símbolos s_1 se encuentran en toda la cinta cuando se le requiera.

En una *unidad de tiempo* de la máquina, ocurren los siguientes pasos en el orden en el que se enuncian.

Verificación de condición de detención. El cabezal obtiene el símbolo de la celda sobre la que se encuentra. Si no encuentra ningún símbolo, éste indica que se encuentra en una celda en blanco. Después busca si existe alguna instrucción que diga qué hacer en el estado actual de la máquina sobre el símbolo obtenido. Si existe se ejecuta el siguiente paso, si no la máquina se detiene.

Lectura. El cabezal lee la instrucción obtenida. Identifica si la instrucción necesita información de un conjunto de números naturales o no. En caso de que sí, lee los estados por los cuales se puede modificar. Además obtiene el número de símbolos s_1 que se encuentran en toda la cinta. En caso contrario, lee el estado por el cual hay que cambiar y distingue de entre los siguientes casos: en caso de que deba modificar el símbolo de la celda, lee el símbolo por el cual hay que hacerlo; en caso contrario, no busca nada e indica el movimiento que tiene que realizar el cabezal, siendo una celda a la derecha o una celda a la izquierda los movimientos permitidos.

Búsqueda. Busca y posteriormente guarda el estado (o los estados) obtenido(s) en el paso anterior. En caso de que tenga que modificar el símbolo obtenido, busca y guarda el símbolo por el cual se debe cambiar. En caso contrario ignora esto último y se ejecuta el siguiente paso.

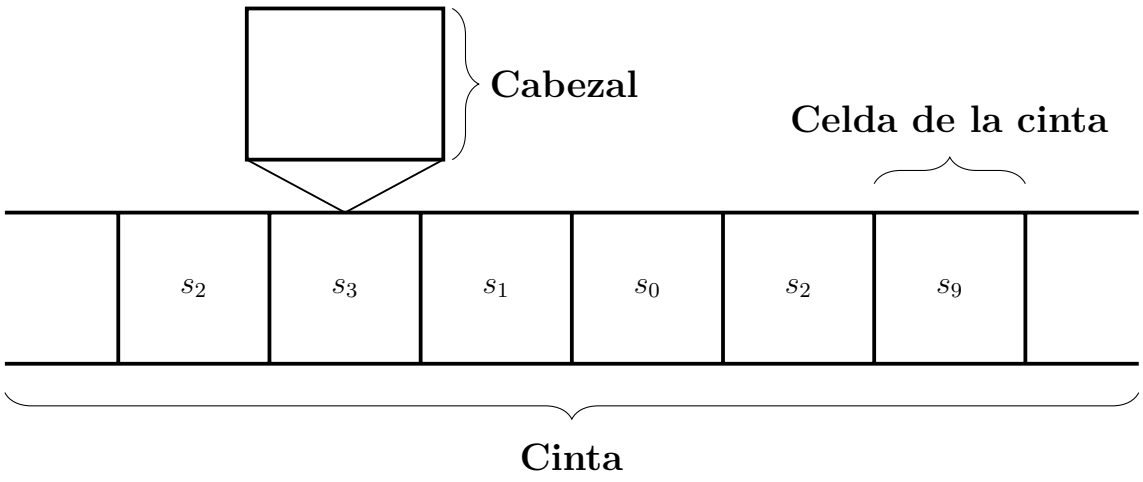

Figura 1.1: Máquina de Turing

Solicitud de información. En caso de que se solicite información, el cursor responde a la pregunta si el número de símbolos s_1 obtenido anteriormente pertenece a un conjunto de números naturales determinado y obtiene una respuesta afirmativa o negativa. En caso contrario, se ejecuta el paso siguiente.

Modificación, movimiento o elección. Si la instrucción solicita información se usa la respuesta de la instrucción anterior y dependiendo a esto elige el estado por el cual hay que modificar. En caso de contrario se distinguen los siguientes casos: en caso de que la instrucción dicte que se debe modificar el símbolo, el cabezal reemplaza el símbolo que se encuentra en la celda por el símbolo que encontró en el paso anterior y permanece posado sobre la misma celda; en caso contrario, el cabezal se mueve una celda a la derecha o una celda a la izquierda según la instrucción se lo haya indicado, sin modificar el símbolo de la celda sobre la cual se encontraba.

Actualización de estado. Si la instrucción solicitó información se reemplaza el estado actual de la máquina por el estado obtenido en el paso Modificación, movimiento o elección; en caso contrario, se reemplaza el estado actual de la máquina por el estado obtenido en el paso Búsqueda. Se ejecuta el paso Verificación de condición de parada.

Formalmente, usamos cantidades finitas de símbolos tomados de conjuntos determinados que representarán los estados de la máquina, los símbolos que se pueden escribir sobre la cinta y los símbolos de movimiento, respectivamente. Con dichos símbolos podremos describir las instrucciones de la máquina, el contenido de la cinta junto con el estado de la máquina en una unidad de tiempo determinada y el cómputo de la máquina en una entrada determinada.

Definimos los conjuntos de símbolos, $\mathbf{Q} := \{q_1, q_2, q_3, ...\}$ que se usarán para representar los estados de la máquina, $\mathbf{S} := \{s_0, s_1, s_2, ...\}$ para representar los símbolos de escritura o letras de la máquina. Además usaremos los símbolos R y L para denotar el movimiento derecho e izquierdo del cabezal, respectivamente. A las sucesiones finitas de símbolos tomados de los conjuntos anteriores les llamaremos **expresiones**.

Por conveniencia, y por fines prácticos, al momento de describir una sucesión de símbolos en la cinta, si una celda está en blanco, en símbolos representamos esto con el símbolo s_0, que para simplificar escribimos como B. También, al símbolo s_1 lo simplificamos con el símbolo 1[1].

Ahora bien, nos interesa decir qué entendemos por instrucciones de la máquina en base a expresiones. Para ello sólo consideramos aquellas que constan de cuatro elementos como las que se muestran a continuación; incluimos también la interpretación de los símbolos que las conforman.

Sean $i, k, l, j, r \in \omega$ con $i, l, r \geq 1$ y $A \subseteq \omega$.

Expresión que denota la instrucción	**Interpretación**
$q_i s_k s_j q_l$	q_i es el estado interno de la máquina; el cabezal de la máquina señala una celda que tiene al símbolo s_k; el cabezal cambia éste símbolo por s_j; la máquina cambia al estado interno q_l
$q_i s_k R q_l$	q_i es el estado interno de la máquina; el cabezal de la máquina señala una celda que tiene al símbolo s_k; el cabezal se mueve una celda a la derecha; la máquina cambia al estado interno q_l
$q_i s_k L q_l$	q_i es el estado interno de la máquina; el cabezal de la máquina señala una celda que tiene al símbolo s_k; el cabezal se mueve una celda a la izquierda; la máquina cambia al estado interno q_l

[1]No consideramos a 1 aritméticamente sino sólo como símbolo.

$q_i s_k q_l q_r$	q_i es el estado interno de la máquina; el cabezal de la máquina señala una celda que tiene al símbolo s_k; el cabezal obtiene el número de símbolos 1 hay en la cinta; si dicho número pertenece a A, el estado interno de la máquina cambia a q_l, en caso contrario la máquina cambia a q_r

Únicamente a este tipo de expresiones les llamamos **cuádruplas**. A las cuádruplas del último tipo descrito las llamaremos **cuádruplas de tipo oráculo respecto a** A.

Representamos a la máquina descrita con anterioridad como un conjunto finito de cuádruplas con una condición particular.

Definición 1.1.1. (Máquina de Turing)

(i) Una **máquina de Turing** es un conjunto finito no vacío de cuádruplas que no tiene dos cuádruplas distintas que coinciden en sus dos primeros símbolos.

(ii) Dado un conjunto $A \subseteq \omega$, una **máquina de Turing con oráculo** A es una máquina de Turing que contiene al menos una cuádrupla de tipo oráculo respecto a A.

Es importante mencionar que existen otro tipo de definiciones de máquinas de Turing. Por ejemplo hay aquellas que tienen más cintas o más cabezales o aquellas que de un paso a otro permiten realizar dos acciones a la vez o bien aquellas que sólo son potencialmente infinitas en una sola dirección. Todas estas definiciones son «equivalentes», en el sentido en el que todos estos modelos de cómputo *calculan* las mismas funciones.

La razón por la que usamos el modelo de cómputo descrito con anterioridad se debe a que es más sencillo explicarlo que los demás y también que en algunas ocasiones no tenemos que preocuparnos si en algún momento dado necesitamos agregar más celdas en alguna dirección.

Dicho esto, procedemos ahora a decir cómo entenderemos los símbolos que se encuentran en la cinta en un momento dado en base a expresiones. Para ello consideraremos aquellas que representan el contenido de la cinta y que además nos indiquen el estado interno de la máquina en un determinado instante.

Definición 1.1.2.

(i) Una **descripción instántanea** α es una expresión que contiene un único símbolo q_i, no contiene R ni L y es tal que q_i no es el último símbolo de ésta expresión.

(ii) Si **T** es una máquina de Turing y α es una descripción instántanea, decimos que α es una **descripción instántanea de T** si y sólo si el símbolo q_i que está en α pertenece también a alguna cuádrupla de **T**.

(iii) Sean **T** una máquina de Turing y una descripción instántanea de **T**, $\alpha = Pq_is_jQ$ para algunas expresiones P, Q posiblemente vacías. Diremos que q_i es el **estado de T en** α y que s_j es el **símbolo escaneado por T en** α. La expresión que se obtiene al remover a q_i de α se denomina la **sucesión de símbolos de la cinta de T en** α.

(iv) Sea **T** una máquina de Turing y sea α una descripción instántanea de **T**, denotaremos por $\langle\alpha\rangle$ al número de símbolos 1 que ocurren en α.

Ahora diremos cómo representar un cómputo de una máquina de Turing con expresiones.

Para esto nos importará indicar cuándo de dos descripciones instántaneas de una máquina de Turing **T** una se sigue de la otra tras aplicar una cuádrupla de **T**.

Definición 1.1.3. *(a)* Sean **T** una máquina de Turing y α, β descripciones instántaneas de **T**. Decimos que β **ocurre después de** α **respecto a T**, lo cual denotamos por $\alpha \to \beta$ (**T**) (o cuando no sea necesario precisar a **T**, $\alpha \to \beta$) si y sólo si alguno de los siguientes casos ocurre:

(i) Existen expresiones P y Q, posiblemente vacías, tal que

$$\alpha \text{ es } Pq_is_jQ \qquad \beta \text{ es } Pq_ls_kQ$$

y la cuádrupla $q_is_js_kq_l$ pertenece a **T**.

(ii) Existen expresiones P y Q, posiblemente vacías, tal que

$$\alpha \text{ es } Pq_is_js_kQ \qquad \beta \text{ es } Ps_jq_ls_kQ$$

y la cuádrupla $q_is_jRq_l$ pertenece a **T**.

(iii) Existen expresiones P y Q, posiblemente vacías, tal que

$$\alpha \text{ es } s_jq_is_kQ \qquad \beta \text{ es } Pq_ls_js_kQ$$

y la cuádrupla $q_is_kLq_l$ pertenece a **T**.

(iv) Existe una expresión P, posiblemente vacía, tal que

$$\alpha \text{ es } Pq_is_j \qquad \beta \text{ es } Ps_jq_ls_0$$

y la cuádrupla $q_is_jRq_l$ pertenece a **T**.

(v) Existe una expresión Q, posiblemente vacía, tal que

$$\alpha \text{ es } q_is_jQ \qquad \beta \text{ es } q_ls_0s_jQ$$

y la cuádrupla $q_is_jLq_l$ pertenece a **T**.

(b) Sean $A \subseteq \omega$, $\mathbf{T}$ una máquina de Turing con oráculo A y α, β descripciones instántaneas de $\mathbf{T}$. Decimos que β **ocurre después de** α **respecto a T y respecto al oráculo** A, lo cual denotamos por $\alpha \to_A \beta$ $(\mathbf{T})$ (o cuando no sea necesario precisar a $\mathbf{T}$, $\alpha \to_A \beta$) si y sólo si

(vi) Existen expresiones P y Q, posiblemente vacías, tal que

$$\alpha \text{ es } Pq_is_kQ \quad , \quad \beta \text{ es } Pq_ls_kQ \quad \text{y} \quad \langle\alpha\rangle \in A$$

o bien

$$\alpha \text{ es } Pq_is_kQ \quad , \quad \beta \text{ es } Pq_rs_kQ \quad \text{y} \quad \langle\alpha\rangle \notin A$$

y la cuádrupla $q_is_kq_lq_r$ pertenece a $\mathbf{T}$.

Algunas observaciones inmediatas de las definiciones anteriores son las siguientes:

Observación. *(i)* Dado $A \subseteq \omega$, si $\mathbf{T}$ es una máquina de Turing (con oráculo A) y α, β, γ son descripciones instántaneas de $\mathbf{T}$ tales que $\alpha \to \beta$ $(\mathbf{T})$, $(\alpha \to_A \beta$ $(\mathbf{T}))$, y $\alpha \to \gamma$ $(\mathbf{T})$, $(\alpha \to_A \gamma$ $(\mathbf{T}))$, entonces $\beta = \gamma$.

(ii) Dado $A \subseteq \omega$, si $\mathbf{T}$ y $\mathbf{T}'$ son máquinas de Turing (con oráculo A) tales que $\mathbf{T} \subseteq \mathbf{T}'$ y α, β son descripciones instántaneas de $\mathbf{T}$ tales que $\alpha \to \beta$ $(\mathbf{T})$, $(\alpha \to_A \beta$ $(\mathbf{T}))$, entonces α, β son descripciones instántaneas de $\mathbf{T}'$ tales que $\alpha \to \beta$ $(\mathbf{T}')$, $(\alpha \to_A \beta$ $(\mathbf{T}'))$.

La intención ahora será considerar tuplas finitas de descripciones instántaneas de una máquina de Turing $\mathbf{T}$, donde la primera descripción instántanea de la tupla tendrá una forma particular.

Definición 1.1.4.

(a) Dada una máquina de Turing $\mathbf{T}$, un **cómputo de T** es una tupla finita no vacía $(\alpha_1, \alpha_2, ..., \alpha_k)$ de descripciones instántaneas de $\mathbf{T}$ tal que:

(i) Para cada $1 \leq i < k$, $\alpha_i \to \alpha_{i+1}$ $(\mathbf{T})$.

(ii) No existe una descripción instántanea β de $\mathbf{T}$ tal que $\alpha_k \to \beta$ $(\mathbf{T})$.

En tal caso, diremos que α_k es el **resultado de** α_1 **respecto a T** y esto lo denotamos por $Res_{\mathbf{T}}(\alpha_1) = \alpha_k$. Y decimos también que el cómputo **inicia** en α_1.

(b) Dados $A \subseteq \omega$ y una máquina de Turing $\mathbf{T}$ con oráculo A, un A**-cómputo de T** es una tupla finita no vacía $(\alpha_1, \alpha_2, ..., \alpha_k)$ de descripciones instántaneas de $\mathbf{T}$ para la cual:

(i) Existen dos conjuntos $M, N \subseteq \{1, ..., k-1\}$ tal que $|M| \geq 1$, $M \cap N = \emptyset$ y $M \cup N = \{1, ..., k-1\}$ de tal modo que si $i \in M$ entonces $\alpha_i \to_A \alpha_{i+1}$ $(\mathbf{T})$ y si $j \in N$ entonces $\alpha_j \to \alpha_{j+1}$ $(\mathbf{T})$.

(ii) No existe una descripción instántanea β de $\mathbf{T}$ tal que $\alpha_k \rightarrow_A \beta$ ($\mathbf{T}$) ó $\alpha_k \rightarrow \beta$ ($\mathbf{T}$).

En tal caso, diremos que α_k es el **resultado de** α_1 **respecto a T con oráculo** A y esto lo denotamos por $Res^A_{\mathbf{T}}(\alpha_1) = \alpha_k$. Y decimos también que el cómputo **inicia** en α_1.

Dado que consideraremos únicamente cómputos cuyas entradas sean números naturales, o bien tuplas de números naturales, a cada número natural $n \in \omega$ le asociamos la expresión $\overline{n} = \underbrace{1 \cdots 1}_{\text{n+1-veces}}$ y a cada tupla finita no vacía de números naturales $(n_1, n_2, ..., n_k) \in \omega^k$ le asociamos la expresión $\overline{(n_1, n_2, ..., n_k)} = \overline{n_1} B \overline{n_2} B \ldots \overline{n_k}$.

Definición 1.1.5. Dados $n \in \omega$ con $n \geq 1$, $A \subseteq \omega$ y una máquina de Turing $\mathbf{T}$, posiblemente con oráculo A, definimos la **función parcial n-aria inducida por T** como la función parcial $\psi^n_{\mathbf{T}} : \omega^n \nrightarrow \omega$ o, si $\mathbf{T}$ es con oráculo A, **la función parcial n-aria inducida por T con oráculo** A como la función parcial $\psi^{n,A}_{\mathbf{T}} : \omega^n \nrightarrow \omega$, tal que:

$$\psi^n_{\mathbf{T}}(x_1, ..., x_n) = \begin{cases} \langle Res_{\mathbf{T}}(\alpha_1) \rangle & \text{si} \quad \text{existe un cómputo de } \mathbf{T} \text{ que inicia en } \alpha_1 = q_1\overline{(x_1, ..., x_n)} \\ indefinida & \text{si} \quad \text{ocurre cualquier otro caso} \end{cases}$$

O en caso de que $\mathbf{T}$ tenga oráculo A:

$$\psi^{n,A}_{\mathbf{T}}(x_1, ..., x_n) = \begin{cases} \langle Res^A_{\mathbf{T}}(\alpha_1) \rangle & \text{si} \quad \text{existe un } A\text{-cómputo de } \mathbf{T} \text{ que inicia en } \alpha_1 = q_1\overline{(x_1, ..., x_n)} \\ indefinida & \text{si} \quad \text{ocurre cualquier otro caso} \end{cases}$$

Para fines prácticos, a $\psi^1_{\mathbf{T}}$ la denotamos únicamente como $\psi_{\mathbf{T}}$ y a $\psi^{1,A}_{\mathbf{T}}$ como $\psi^A_{\mathbf{T}}$.

Definición 1.1.6. Dados $n \in \omega$ con $n \geq 1$, $A \subseteq \omega$ y $f : \omega^n \nrightarrow \omega$ una función parcial. Decimos que,

(i) f es **parcialmente Turing-computable** si y sólo si existe una máquina de Turing $\mathbf{T}$ tal que $\psi^n_{\mathbf{T}} = f$.

(ii) f es **parcialmente A-Turing-computable** si y sólo si existe una máquina de Turing $\mathbf{T}$ con oráculo A tal que $\psi^{n,A}_{\mathbf{T}} = f$.

(iii) f es **Turing-computable** si y sólo si f es parcialmente Turing-computable y dom $f = \omega^n$.

(iv) f es A-**Turing-computable** si y sólo si f es parcialmente A-Turing-computable y dom $f = \omega^n$.

De ahora en adelante, en lugar de decir que una función es parcialmente Turing-computable diremos únicamente que es parcialmente computable, o bien si una función es Turing-computable diremos que es computable y de la misma forma con las versiones relativas a un conjunto $A \subseteq \omega$.

Esto no se debe a una mera convención en los términos utilizados, como mencionamos en la introducción, existe una definición informal de algoritmo, sobre la cual reposa la noción de que una función sea «computable», la cual se refiere a que exista un algoritmo que siempre que tome un elemento en el dominio de la función, éste devuelva el valor de la función valuada en dicho elemento. A lo largo de la definición de una máquina de Turing hemos mencionado como éste objeto abstracto puede representar una forma cercana de la idea de algoritmo. Esto último, a pesar de ser explicado con suficiente detalle, no es una demostración de que la idea de algoritmo es capturada completamente por una máquina de Turing, más bien es una creencia conocida *Tesis de Church-Turing*.

En pocas palabras éste postulado nos dice que toda función computable en el sentido informal es computable por una máquina de Turing y viceversa. En este trabajo admitiremos la tesis de Church-Turing y algunas veces, por comodidad, demostraremos que una función es computable simplemente describiendo un algoritmo que cómputa la función asumiendo que uno puede construir una máquina de Turing basada en dicho algoritmo que cómpute la función; a este método de demostración lo llamamos *prueba por tesis de Church-Turing*.

El hecho de que podamos realizar este tipo de prueba no debe hacer que despreciemos el estudio de las máquinas de Turing y posteriormente de las máquinas de acceso aleatorio, ya que éstos son objetos formales, denotados en lenguaje matemático, que nos ayudan a darle solidez no sólo a los resultados que queremos estudiar si no a toda la Teoría de la Computabilidad.

Sin más, a continuación mostraremos algunos ejemplos de funciones computables usando máquinas de Turing.

Ejemplo 1.1.1. La función $S : \omega \to \omega$ tal que $S(x) = x + 1$ es computable.

Demostración. En efecto, considere la máquina de Turing $\mathbf{T}$ cuya única cuádrupla es:

$$q_1 1 R q_2$$

Afirmamos que para todo $m \in \omega$, $\psi_{\mathbf{T}}(m) = S(m)$.
En efecto, sea $m \in \omega$,

$$\begin{aligned} q_1\overline{m} &= q_1 1^{m+1} \\ &= q_1 1 1^m \\ &\to 1 q_2 1^m \end{aligned}$$

Por lo tanto,

$$\begin{aligned}\psi_{\mathbf{T}}(m) &= \langle 1q_2 1^m \rangle \\ &= m+1 \\ &= S(m)\end{aligned}$$

Concluimos que $\psi_{\mathbf{T}} = S$ y por lo tanto que S es computable. □

Ejemplo 1.1.2. La función $f : \omega \times \omega \to \omega$ tal que $f(x, y) = x + y$ es computable.

Demostración. Sea $\mathbf{T}$ la máquina de Turing cuyas cuádruplas son:

$q_1 1 B q_1$	Borra un símbolo 1
$q_1 B R q_2$ $q_2 1 R q_2$ $q_2 B R q_3$	Ignora los demás símbolos 1 e identifica el primer símbolo B.
$q_3 1 B q_3$	Borra un símbolo 1.

Afirmamos que para cualesquiera $n_1, n_2 \in \omega$, $f(n_1, n_2) = \psi^2_{\mathbf{T}}(n_1, n_2)$.
En efecto, sean $n_1, n_2 \in \omega$,

$$\begin{aligned}q_1 \overline{n_1} B \overline{n_2} &= q_1 1^{n_1+1} B 1^{n_2+1} \\ &= q_1 1 1^{n_1} B 1^{n_2+1} \\ &\to q_1 B 1^{n_1} B 1^{n_2+1} \\ &\to B q_2 1^{n_1} B 1^{n_2+1} \\ &\to \dots \\ &\to B 1^{n_1} q_2 B 1^{n_2+1} \\ &\to B 1^{n_1} B q_3 1^{n_2+1} \\ &= B 1^{n_1} B q_3 1 1^{n_2} \\ &\to B 1^{n_1} B q_3 B 1^{n_2}\end{aligned}$$

Por lo tanto,

$$\begin{aligned}\psi^2_T(n_1, n_2) &= \langle B 1^{n_1} B q_3 B 1^{n_2} \rangle \\ &= n_1 + n_2 \\ &= f(n_1, n_2)\end{aligned}$$

Concluimos que $\psi^2_{\mathbf{T}} = f$ y por lo tanto que f es computable. □

De la misma forma, podemos demostrar que funciones más interesantes son computables. Sin embargo, el ejemplo anterior nos da una idea de lo tedioso que puede llegar a ser esto. Es por ello que a continuación mostraremos un modelo de cómputo más flexible y del mismo «poder», en el sentido de que cómputa las mismas

funciones que el recién mostrado. Con este modelo demostraremos resultados más importantes para la teoría que el únicamente verificar casos particulares de funciones computables.

1.1.2. Máquinas de Acceso Aleatorio

Gráficamente, una *máquina de acceso aleatorio* es una *máquina* que consiste de una *cinta* bidimensional dividida en celdas del mismo tamaño que denominaremos *registros* y de un *controlador* conectado a la cinta.

La cinta será infinita únicamente en una dirección y cada registro puede contener un número natural. El controlador contiene una cantidad finita y ordenada de instrucciones que llamaremos *programa* y que le permite obtener y modificar el contenido de cualquier registro. Dentro de las instrucciones posibles, tendremos a aquellas que nos permiten obtener y enviar contenidos a cualquier registro cuyo indice depende del contenido de cualquier otro registro.

En una *unidad de tiempo* de la máquina ocurren los siguientes pasos:

Verificación de condición de parada. El controlador verifica en un contador el número de la instrucción a ejecutar. Si este número es menor igual al número de la última instrucción, se ejecuta el siguiente paso, si no la máquina se detiene.

Lectura de instrucción. El controlador identifica o consulta el contenido de algún o algunos registros para obtener el número del o de los registros cuyo contenido es necesario para que la instrucción pueda ejecutarse. Además identifica si la instrucción necesita información extra.

Obtención de contenido. El controlador obtiene el contenido del o de los registros del paso anterior. En caso de que sean dos contenidos, los obtiene en el orden en el que la instrucción lo requiere.

Solicitud de información. Si la instrucción necesita información extra entonces el controlador la solicita a un conjunto de números naturales y en caso contrario se ejecuta el siguiente paso.

Modificación de contenido. En el controlador se realizan la(s) modificacion(es) declaradas por la instrucción sobre el (los) contenido(s) obtenido(s), también usando la información obtenida en el caso anterior en caso de ser necesario.

Modificación de cinta. El controlador modifica los registros que indica la instrucción conforme al orden que ésta dicta y con los números obtenidos por el paso anterior. Aumenta en uno el contador de instrucciones y vamos al paso Verificación de condición de parada.

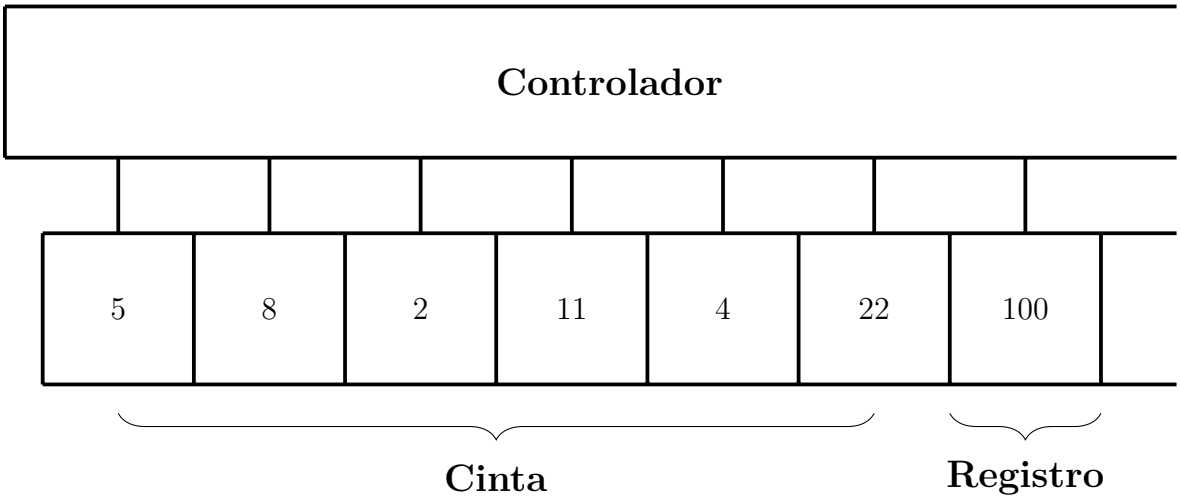

Figura 1.2: Máquina de Acceso Aleatorio

El hecho de que en una unidad de tiempo podamos acceder al contenido de cualquier registro da significado a *acceso aleatorio* en el nombre de estas máquinas. En contraste con el *acceso secuencial* con el que se comportan las máquinas de Turing, que para acceder al contenido de una celda de la cinta, debemos acceder al contenido de las que se encuentran a la derecha o izquierda de éstas.

Dicho esto, pasamos a describir formalmente el comportamiento de las máquinas de acceso aleatorio. Para ello consideramos un conjunto de símbolos $\mathbf{R} := \{r_n : n \in \mathbb{N}\}$ que servirán para referirnos a los contenidos de los registros e $\mathbf{I} := \{I_n : n \in \mathbb{N}\}$ que usaremos para referirnos a instrucciones de la máquina.

A continuación damos un listado formal de las instrucciones permitidas por la máquina:

Instrucciones cero. Para cada $n \in \mathbb{N}$, la instrucción Z_n que indica que el contenido r_n debe cambiarse por el número 0. También escribimos esta instrucción como $r_n \leftarrow 0$.
Dado $n \in \mathbb{N}$, sea $Z_n : \omega^{<\mathbb{N}} \times \mathbb{N} \to \omega^{<\mathbb{N}} \times \mathbb{N}$ la función tal que:

$$Z_n((x_1, ..., x_k), i) = \begin{cases} ((x_1, ..., \underbrace{0}_{\text{posición } n}, ..., x_k), i+1) & \text{si } n \leq k \\ ((x_1, ..., x_k, 0, ..., \underbrace{0}_{\text{posición } n}), i+1) & \text{si } k < n \end{cases}$$

Instrucciones sucesor. Para cada $n \in \mathbb{N}$, la instrucción S_n que indica que el contenido r_n, debe cambiarse por $r_n + 1$. También escribimos esta instrucción como $r_n \leftarrow r_n + 1$.
Dado $n \in \mathbb{N}$, sea $S_n : \omega^{<\mathbb{N}} \times \mathbb{N} \to \omega^{<\mathbb{N}} \times \mathbb{N}$ la función tal que:

$$S_n((x_1, ..., x_k), i) = \begin{cases} ((x_1, ..., x_n + 1, ..., x_k), i + 1) & \text{si } n \leq k \\ ((x_1, ..., x_k, 0, ..., 0, \underbrace{1}_{\text{posición } n}), i + 1) & \text{si } k < n \end{cases}$$

Instrucciones decremento. Para cada $n \in \mathbb{N}$, la instrucción D_n que indica que r_n, debe cambiarse por $r_n - 1$ si $r_n \geq 1$ y por 0 si $r_n = 0$. También escribimos esta instrucción como $r_n \leftarrow r_n \dot{-} 1$.
Dado $n \in \mathbb{N}$, sea $D_n : \omega^{<\mathbb{N}} \times \mathbb{N} \to \omega^{<\mathbb{N}} \times \mathbb{N}$ la función tal que:

$$D_n((x_1, ..., x_k), i) = \begin{cases} ((x_1, ..., x_n - 1, ..., x_k), i + 1) & \text{si } n \leq k \ \wedge \ x_n > 0 \\ ((x_1, ..., 0, ..., x_k), i + 1) & \text{si } n \leq k \ \wedge \ x_n = 0 \\ ((x_1, ..., x_k, 0, ..., 0, \underbrace{0}_{\text{posición } n}), i + 1) & \text{si } k < n \end{cases}$$

Instrucciones de transferencia. Para cada $m, n \in \mathbb{N}$, la instrucción $T_{m,n}$ que indica que r_n debe cambiarse por r_m. También escribimos esta instrucción como $r_n \leftarrow r_m$.
Dados $m, n \in \mathbb{N}$, sea $T_{m,n} : \omega^{<\mathbb{N}} \times \mathbb{N} \to \omega^{<\mathbb{N}} \times \mathbb{N}$ la función tal que:

$$T_{m,n}((x_1, ..., x_k), i) = \begin{cases} ((x_1, ..., \underbrace{x_m}_{\text{posición } n}, ..., x_m, ..., x_k), i + 1) & \text{si } n \leq m \leq k \\ ((x_1, ..., x_m, ..., \underbrace{x_m}_{\text{posición } n}, ..., x_k), i + 1) & \text{si } m \leq n \leq k \\ ((x_1, ..., x_m, ..., x_k, 0, ..., 0, \underbrace{x_m}_{\text{posición } n}), i + 1) & \text{si } m \leq k < n \\ ((x_1, ..., \underbrace{0}_{\text{posición } n}, ..., x_k), i + 1) & \text{si } n \leq k < m \\ ((x_1, ..., x_k), i + 1) & \text{si } k < m, n \end{cases}$$

Instrucciones de salto. Para cada $m, n, k \in \mathbb{N}$, la instrucción $J_{m,n,k}$ que indica que si r_m y r_n son iguales entonces el contador que dice cuál debe ser la instrucción que debe ejecutarse pasa a ser el número k, en otro caso dicho contador sólo aumenta en uno.

Dados $m, n, s \in \mathbb{N}$, sea $J_{m,n,s} : \omega^{<\mathbb{N}} \times \mathbb{N} \to \omega^{<\mathbb{N}} \times \mathbb{N}$ la función tal que:

$$J_{m,n,s}((x_1, ..., x_k), i) = \begin{cases} ((x_1, ..., x_k), s) & \text{si } (x_m = x_n \ \wedge \ n, m \leq k) \ \text{ ó} \\ & (x_m = 0 \ \wedge \ m \leq k < n) \ \text{ ó} \\ & (x_n = 0 \ \wedge \ n \leq k < m) \ \text{ ó} \\ & k < m, n \\ ((x_1, ..., x_k), i + 1) & \text{si ocurre otro caso} \end{cases}$$

Escribimos esta instrucción como:

si $r_m = r_n$ **entonces**
  ir a instrucción número s
en otro caso
  ir a instrucción siguiente

Instrucciones de carga. Para cada $m, n \in \mathbb{N}$, la instrucción $L_{m,n}$ que indica que r_n debe ser sustituido por el contenido del registro cuyo índice es r_m. A esta instrucción la escribimos como $r_n \leftarrow r_{r_m}$.
Dados $m, n \in \mathbb{N}$, sea $L_{m,n} : \omega^{<\mathbb{N}} \times \mathbb{N} \to \omega^{<\mathbb{N}} \times \mathbb{N}$ la función tal que:

$$L_{m,n}((x_1, ..., x_k), i) = \begin{cases} ((x_1, x_2, ..., \underbrace{x_j}_{\text{posición } n}, ..., x_k), i + 1) & \text{si } m, n \leq k \wedge \\ & 0 < j = x_m \leq k \\ ((x_1, ..., x_k), i + 1) & \text{si } m, n \leq k \\ & \wedge \ x_m = 0 \\ ((x_1, x_2, ..., \underbrace{0}_{\text{posición } n}, ..., x_k), i + 1) & \text{si } m, n \leq k \\ & \wedge \ k < x_m \\ ((x_1, x_2, ..., x_k, 0, ..., 0, \underbrace{x_j}_{\text{posición } n}), i + 1) & \text{si } 0 < m \leq k \\ & \wedge \ 0 < x_m = j \leq k \\ & \wedge \ k < n \\ ((x_1, ..., x_k), i + 1) & \text{si ocurre otro caso} \end{cases}$$

Instrucciones de almacenamiento. Para cada $m, n \in \mathbb{N}$, la instrucción $G_{m,n}$ que indica que el contenido del registro cuyo índice es r_m debe ser sustituido por r_n. A esta instrucción la escribimos como $r_{r_m} \leftarrow r_n$.

Dados $m, n \in \mathbb{N}$, sea $G_{m,n} : \omega^{<\mathbb{N}} \times \mathbb{N} \to \omega^{<\mathbb{N}} \times \mathbb{N}$ la función tal que:

$$G_{m,n}((x_1, ..., x_k), i) = \begin{cases} ((x_1, x_2, ..., \underbrace{x_n}_{\text{posición } j}, ..., x_k), i+1) & \text{si } m, n \leq k \wedge \\ & 0 < x_m = j \leq k \\ ((x_1, ..., x_k), i+1) & \text{si } m, n \leq k \\ & \wedge\ x_m = 0 \\ ((x_1, x_2, ..., x_k, 0, ..., 0, \underbrace{x_n}_{\text{posición } j}), i+1) & \text{si } m, n \leq k \wedge \\ & k < x_m = j \\ ((x_1, ..., \underbrace{0}_{\text{posición } j}, ..., x_k), i+1) & \text{si } m \leq k < n \wedge \\ & 0 < x_m = j \leq k \\ ((x_1, ..., x_k), i+1) & \text{si ocurre otro caso} \end{cases}$$

Instrucciones de suma. La instrucción $Q_{m,n}$ que indica a r_m lo debemos sustituir por la suma de r_n y r_m. A esta instrucción la denotamos por $r_n \leftarrow r_n + r_m$.
Dados $m, n \in \mathbb{N}$, sea $Q_{m,n} : \omega^{<\mathbb{N}} \times \mathbb{N} \to \omega^{<\mathbb{N}} \times \mathbb{N}$ la función tal que:

$$Q_{m,n}((x_1, ..., x_k), i) = \begin{cases} ((x_1, ..., \underbrace{x_n + x_m}_{\text{posición } n}, ..., x_m, ..., x_k), i+1) & \text{si } n \leq m \leq k \\ ((x_1, ..., x_m, ..., \underbrace{x_n + x_m}_{\text{posición } n}, ..., x_k), i+1) & \text{si } m \leq n \leq k \\ ((x_1, ..., x_m, ..., x_k, 0, ..., 0, \underbrace{x_m}_{\text{posición } n}), i+1) & \text{si } m \leq k < n \\ ((x_1, ..., x_k), i+1) & \text{si ocurre otro caso} \end{cases}$$

Instrucciones oráculo. Dado un conjunto de números naturales $A \subseteq \omega$, la instrucción O_n^A que indica que si r_n pertenece al conjunto A entonces dicho contenido se modifica por el número 1, en caso contrario se modifica por el número 0.

Dado $n \in \mathbb{N}$, sea $O_n^A : \omega^{<\mathbb{N}} \times \mathbb{N} \to \omega^{<\mathbb{N}} \times \mathbb{N}$ la función tal que:

$$O_n^A((x_1, ..., x_k), i) = \begin{cases} ((x_1, ..., \underbrace{\chi_A(x_n)}_{\text{posición } n}, ..., x_k), i+1) & \text{si } n \leq k \\ ((x_1, ..., x_k, 0, ..., 0, \underbrace{\chi_A(x_n)}_{\text{posición } n}), i+1) & \text{si } k < n \end{cases}$$

También escribimos esta instrucción como:

si $r_n \in A$ **entonces**
$r_n \leftarrow 1$
en otro caso
$r_n \leftarrow 0$

A continuación una breve definición:

Definición 1.1.7. *(i)* Al conjunto

$$\begin{aligned} \mathbf{INST} := \ & \{Z_n : n \in \mathbb{N}\} \cup \{S_n : n \in \mathbb{N}\} \cup \{D_n : n \in \mathbb{N}\} \cup \{T_{m,n} : m, n \in \mathbb{N}\} \cup \\ & \cup \{L_{m,n} : m, n \in \mathbb{N}\} \cup \{J_{m,n,s} : n, m, s \in \mathbb{N}\} \cup \\ & \cup \{Q_{m,n} : m, n \in \mathbb{N}\} \cup \{G_{m,n} : m, n \in \mathbb{N}\} \end{aligned}$$

lo llamamos **conjunto de instrucciones RAM**.

(ii) Dado $A \subseteq \omega$, al conjunto

$$\mathbf{INST}^A := \ \mathbf{INST} \cup \{O_n^A : n \in \mathbb{N}\}$$

lo llamamos **conjunto de instrucciones RAM con oráculo** A.

Teniendo claro que entendemos por instrucción y conjunto de instrucciones ya podemos decir qué entenderemos por programa. A continuación damos una definición formal de esto.

Definición 1.1.8. *(i)* Diremos que una tupla finita $\mathbf{P} = (\mathbf{I}_1, ..., \mathbf{I}_k) \in ((\omega^{<\mathbb{N}} \times \mathbb{N})^{\omega^{<\mathbb{N}} \times \mathbb{N}})^{<\mathbb{N}}$ es un **programa RAM** si y sólo si para cada $1 \leq j \leq k$, $\mathbf{I}_j \in \mathbf{INST}$. Denotamos por **PROG** al conjunto de todos los programas RAM.

(ii) Dado $A \subseteq \omega$, diremos que una tupla finita $\mathbf{P}^A = (\mathbf{I}_1, ..., \mathbf{I}_k) \in ((\omega^{<\mathbb{N}} \times \mathbb{N})^{\omega^{<\mathbb{N}} \times \mathbb{N}})^{<\mathbb{N}}$ es un **programa RAM con oráculo** A si y sólo si existen $M, N \subseteq \{1, ..., k\}$ tal que $|M| \geq 1$, $M \cap N = \emptyset$ y $M \cup N = \{1, ..., k\}$ tal que si $j \in M$ entonces $\mathbf{I}_j = O_n^A$ para alguna $n \in \mathbb{N}$ y si $s \in N$ entonces $\mathbf{I}_s \in \mathbf{INST}^A$. Denotamos por $\mathbf{PROG}^A$ al conjunto de todos los programas RAM on oráculo A.

Un pequeño paréntesis es que al momento de dar un programa RAM, posiblemente con oráculo, por lo regular enlistaremos las instrucciones de éste de la manera informal de modo que sea más sencillo entender la instrucción.

Ahora bien, todo programa induce una función de control que hace actuar las funciones instrucción sobre tuplas finitas de números naturales y además lleva un contador sobre qué instrucción se está ejecutando.

Definición 1.1.9. *(i)* Sea $\mathbf{P} = (\mathbf{I}_1, ..., \mathbf{I}_k)$ un programa RAM. Definimos **la función de control de P**, $Ctrl_{\mathbf{P}} : (\omega^{<\mathbb{N}} \times \mathbb{N}) \cup \omega \to (\omega^{<\mathbb{N}} \times \mathbb{N}) \cup \omega$ como la función tal que:

$$Ctrl_{\mathbf{P}}(m) = m$$

$$Ctrl_{\mathbf{P}}((x_1, ..., x_n), r) = \begin{cases} \mathbf{I}_r((x_1, ..., x_n), r) & \text{si } r \leq k \\ x_1 & \text{si } k < r \end{cases}$$

(ii) Dado $A \subseteq \omega$, sea $\mathbf{P}^A = (\mathbf{I}_1, ..., \mathbf{I}_k)$ un programa RAM con oráculo A. Definimos **la función de control de $\mathbf{P}^A$**, $Ctrl_{\mathbf{P}^A} : (\omega^{<\mathbb{N}} \times \mathbb{N}) \cup \omega \to (\omega^{<\mathbb{N}} \times \mathbb{N}) \cup \omega$ como la función tal que:

$$Ctrl_{\mathbf{P}^A}(m) = m$$

$$Ctrl_{\mathbf{P}^A}((x_1, ..., x_n), r) = \begin{cases} \mathbf{I}_r((x_1, ..., x_n), r) & \text{si } r \leq k \\ x_1 & \text{si } k < r \end{cases}$$

Nos interesa saber si tras iterar la función de control de un programa $\mathbf{I}$ en algún momento obtenemos que el valor de dicha función en una tupla específica devuelve un número natural.

Definición 1.1.10. *(i)* Dados $n \in \mathbb{N}$, $(x_1, ..., x_n) \in \omega^n$ y $\mathbf{P}$ un programa RAM, decimos que **P se detiene** en $(x_1, ..., x_n)$, lo cual denotamos por $\mathbf{P}(x_1, ..., x_n) \downarrow$ si y sólo si existe $m \in \omega$ tal que $Ctrl^m_{\mathbf{P}}((x_1, ..., x_n), 1) \in \omega$. En caso de que $\mathbf{P}$ no se detenga en $(x_1, ..., x_n)$ escribiremos $\mathbf{P}(x_1, ..., x_n) \uparrow$.

(ii) Dados $n \in \mathbb{N}$, $(x_1, ..., x_n) \in \omega^n$, $\alpha \in \omega$ y $\mathbf{P}$ un programa RAM, diremos que **P converge a α en** $(x_1, ..., x_n)$, lo cual denotamos por $\mathbf{P}(x_1, ..., x_n) \downarrow \alpha$ si y sólo si $\mathbf{P}$ se detiene en $(x_1, ..., x_n)$ y $\alpha = Ctrl^m_{\mathbf{P}}((x_1, ..., x_n), 1)$, donde $m = \min_{z \in \omega}\{Ctrl^z_{\mathbf{P}}((x_1, ..., x_n), 1) \in \omega\}$.

(iii) Dados $A \subseteq \omega$, $n \in \mathbb{N}$, $(x_1, ..., x_n) \in \omega^n$ y $\mathbf{P}^A$ un programa RAM con oráculo A, decimos que **$\mathbf{P}^A$ se detiene** en $(x_1, ..., x_n)$, lo cual denotamos por $\mathbf{P}^A(x_1, ..., x_n) \downarrow$ si y sólo si existe $m \in \omega$ tal que $Ctrl^m_{\mathbf{P}^A}((x_1, ..., x_n), 1) \in \omega$. En caso de que $\mathbf{P}$ no se detenga en $(x_1, ..., x_n)$ escribiremos $\mathbf{P}^A(x_1, ..., x_n) \uparrow$.

(iv) Dados $A \subseteq \omega$, $n \in \mathbb{N}$, $(x_1, ..., x_n) \in \omega^n$, $\alpha \in \omega$ y $\mathbf{P}^A$ un programa RAM con oráculo A, diremos que **$\mathbf{P}^A$ converge a α en** $(x_1, ..., x_n)$, lo cual denotamos por $\mathbf{P}^A(x_1, ..., x_n) \downarrow \alpha$ si y sólo si $\mathbf{P}^A$ se detiene en $(x_1, ..., x_n)$ y $\alpha = Ctrl^m_{\mathbf{P}^A}((x_1, ..., x_n), 1)$, donde $m = \min_{z \in \omega}\{Ctrl^z_{\mathbf{P}^A}((x_1, ..., x_n), 1) \in \omega\}$.

Definición 1.1.11. *(i)* Dados $n \in \mathbb{N}$, $(x_1, ..., x_n) \in \omega^n$ y $\mathbf{P}$ un programa RAM. Si $\mathbf{P}(x_1, ..., x_n) \downarrow \alpha$ para alguna $\alpha \in \omega$, decimos que $t \in \omega$ es **el tiempo de ejecución de P en** $(x_1, ..., x_n)$ si y sólo si $t = \min_{z \in \omega}\{Ctrl^z_{\mathbf{P}}((x_1, ..., x_n), 1) \in \omega\}$.

(ii) Dados $A \subseteq \omega$, $n \in \mathbb{N}$, $(x_1, ..., x_n) \in \omega^n$ y $\mathbf{P}^A$ un programa RAM con oráculo A. Si $\mathbf{P}^A(x_1, ..., x_n) \downarrow \alpha$ para alguna $\alpha \in \omega$, decimos que y $t \in \omega$ es **el tiempo de ejecución de $\mathbf{P}^A$ en** $(x_1, ..., x_n)$ si y sólo si $t = \min_{z \in \omega}\{Ctrl^z_{\mathbf{P}^A}((x_1, ..., x_n), 1) \in \omega\}$.

En la definición anterior, observemos que el tiempo de ejecución, en caso de que exista, es mayor o igual a 2 ya que se necesita al menos una ejecución de $Ctrl$ para obtener el número de una instrucción mayor a la cantidad de instrucciones del programas y otra ejecución de $Ctrl$ para obtener un número natural.

De forma similar a la definición con máquinas de Turing, todo programa RAM, posiblemente con oráculo A para algún conjunto $A \subseteq \omega$, induce una función n-aria para cada $n \in \mathbb{N}$.

Definición 1.1.12. *(i)* Dados $n \in \mathbb{N}$ y $\mathbf{P}$ un programa RAM, definimos la **función n-aria inducida por P** como la función parcial $\Phi_{\mathbf{P}} : \omega^n \nrightarrow \omega$ tal que:

$$\Phi^n_{\mathbf{P}}(x_1, ..., x_n) = \begin{cases} \alpha & \text{si } \mathbf{P}(x_1, ..., x_n) \downarrow \alpha \\ indefinida & \text{si ocurre cualquier otro caso} \end{cases}$$

(ii) Dados $A \subseteq \omega$, $n \in \mathbb{N}$ y $\mathbf{P}$ un programa RAM con oráculo A, definimos la **función n-aria inducida por $\mathbf{P}^A$** como la función parcial $\Phi_{\mathbf{P}^A} : \omega^n \nrightarrow \omega$ tal que:

$$\Phi^n_{\mathbf{P}^A}(x_1, ..., x_n) = \begin{cases} \alpha & \text{si } \mathbf{P}^A(x_1, ..., x_n) \downarrow \alpha \\ indefinida & \text{si ocurre cualquier otro caso} \end{cases}$$

Para fines prácticos, de ahora en adelante sustituiremos $\Phi^1_{\mathbf{P}}$, $\Phi^{A,1}_{\mathbf{P}}$, por $\Phi_{\mathbf{P}}$, $\Phi^A_{\mathbf{P}}$.

Dicho esto ya podemos decir cuándo una función es computable en el sentido RAM.

Definición 1.1.13. Dados $A \subseteq \omega$, $n \in \mathbb{N}$ y $f : \omega^n \nrightarrow \omega$ una función parcial, decimos que:

(i) f es **parcialmente RAM-computable** si y sólo si existe un programa RAM $\mathbf{P}$ tal que $\Phi^n_{\mathbf{P}} = f$.

(ii) f es **parcialmente A-RAM-computable** si y sólo si existe un programa RAM con oráculo A $\mathbf{P}^A$ tal que $\Phi^n_{\mathbf{P}^A} = f$.

(iii) f es **RAM-computable** si y sólo f es parcialmente RAM-computable y dom $f = \omega^n$.

(iv) f es A**-RAM-computable** si y sólo f es parcialmente A-RAM-computable y dom $f = \omega^n$.

Como ocurre con máquinas de Turing, cuando una función sea (parcialmente A-) RAM-computable, únicamente diremos que esta función es (parcialmente A-) computable. A diferencia del caso anterior, la razón por la que hacemos este cambio no es sólo la tesis de Church-Turing si no también se debe también al hecho de que toda función (parcialmente A-) RAM-computable es (parcialmente A-) Turing-computable y viceversa. Por lo tanto, suponiendo cierta la tesis de Church-Turing y tomando en cuenta el resultado de equivalencia mencionado anteriormente, tenemos que una función es (parcialmente A-) computable en el sentido informal si y sólo si es (parcialmente A-) Turing-computable si y sólo si es (parcialmente A-) RAM-computable.

Unas observaciones importantes e inmediatas son las siguientes:

Observación. Para todo $A \subseteq \omega$ y $n \in \mathbb{N}$, si una función parcial $f : \omega^n \nrightarrow \omega$ es parcialmente computable entonces también es parcialmente A-computable.

Demostración. Sea $\mathbf{P} = (\mathbf{I}_1, ..., \mathbf{I}_k)$ el programa RAM tal que $\Phi^n_{\mathbf{P}} = f$. Sea $\mathbf{I}_{k+1} = O^A_2$. Definimos el programa con oráculo A, $\mathbf{Q}^A = (\mathbf{I}_1, ..., \mathbf{I}_k, \mathbf{I}_{k+1})$. Notemos que el programa $\mathbf{Q}^A$ es el programa $\mathbf{P}$ con una instrucción adicional, a la cual sólo se puede acceder cuando el programa $\mathbf{P}$ está a un sólo paso para detenerse debido a que se encuentra en una tupla que tiene como número de instrucción a $k + 1$. Si $\mathbf{Q}^A$ se detiene en una tupla es porque se debió detener el programa $\mathbf{P}$ en la misma tupla con el mismo resultado ya que la instrucción añadida no afecta el primer elemento de la tupla obtenida por $\mathbf{P}$. Y viceversa, el resultado de $\mathbf{P}$, al no involucrar la nueva instrucción, el resultado de $\mathbf{Q}^A$ debe ser el mismo. Por lo tanto, $\Phi^n_{\mathbf{Q}^A} = f$. Y con esto concluimos que f es parcialmente A-computable. □

Otra observación importante es la siguiente:

Observación. Una función parcial $f : \omega^n \nrightarrow \omega$ es parcialmente computable si y sólo si es $\emptyset$-parcialmente computable.

Demostración. Por la observación anterior, tenemos que si f es parcialmente computable en particular es $\emptyset$-parcialmente computable. Ahora bien, si f es $\emptyset$-parcialmente computable. Sea $\mathbf{Q}^\emptyset$ el programa tal que $\Phi^n_{\mathbf{Q}^\emptyset} = f$. Notemos que para todo $n \in \omega$, $O^\emptyset_n = Z_n$. Por lo tanto, toda instrucción en $\mathbf{Q}^\emptyset$ de la forma $O^\emptyset_n$ para algun $n \in \omega$, la sustituimos por Z_n para formar un programa $\mathbf{P}$ sin instrucciones con oráculo $\emptyset$, únicamente con instrucciones de los demás tipos. Es evidente que $\Phi^n_{\mathbf{P}} = f$. Con esto concluimos que f es parcialmente computable. □

A continuación mostraremos como los ejemplos de funciones anteriormente demostradas vía Turing-computabilidad son fácilmente demostrables ser RAM-computables y por lo tanto computables.

Ejemplo 1.1.3. La función $C : \omega \to \omega$ tal que $C(x) = 0$ es computable.

Demostración. Sea $\mathbf{P} = Z_1$, o bien sea $\mathbf{P}$ el programa con la única instrucción $\mathbf{I}_1 : r_1 \leftarrow 0$. Por lo tanto, dado $n \in \omega$, $Ctrl^2_{\mathbf{P}}(n, 1) = Ctrl_{\mathbf{P}}(Z_1(n, 1)) = Ctrl_{\mathbf{P}}(0, 2) = 0$. De modo que, $\mathbf{P}(n) \downarrow 0$. Por lo tanto $\Phi_{\mathbf{P}} = C$. Con lo cual concluimos que C es computable. □

Ejemplo 1.1.4. La función $S : \omega \to \omega$ tal que $S(x) = x + 1$ es computable.

Demostración. Sea $\mathbf{P} = S_1$, o bien sea $\mathbf{P}$ el programa con la única instrucción $\mathbf{I}_1 : r_1 \leftarrow r_1 + 1$. Entonces, dado $n \in \omega$, $Ctrl^2_{\mathbf{P}}(n, 1) = Ctrl_{\mathbf{P}}(S_1(n, 1)) = Ctrl_{\mathbf{P}}(n + 1, 2) = n + 1$. De modo que, $\mathbf{P}(n) \downarrow n + 1$. Por lo tanto $\Phi_{\mathbf{P}} = S$. Con lo cual concluimos que S es computable. □

Ejemplo 1.1.5. La función $f : \omega \times \omega \to \omega$ tal que $f(x, y) = x + y$ es computable.

Demostración. Sea $\mathbf{P} = Q_{2,1}$. Dados $n, m \in \omega$, tenemos que $Ctrl^2_{\mathbf{P}}((n, m), 1) = Ctrl_{\mathbf{P}}(Q_{2,1}((n, m), 1)) = Ctrl_{\mathbf{P}}((n + m), 2) = n + m$. Por lo tanto, $\Phi^2_{\mathbf{P}}(n, m) = n+m = f(n, m)$, es decir que $\Phi^2_{\mathbf{P}} = f$, de lo que concluimos que f es computable. □

Ejemplo 1.1.6. La función $Pred : \omega \to \omega$ tal que

$$Pred(x) = \begin{cases} x - 1 & \text{si} \quad x \geq 1 \\ 0 & \text{si} \quad x = 0 \end{cases}$$

es computable.

Demostración. Sea $\mathbf{Q} = D_1$. Dado $n \in \omega$, si $n = 0$ entonces $Ctrl^2_{\mathbf{Q}}(n, 1) = Ctrl_{\mathbf{Q}}(D_1(0, 2)) = Ctrl_{\mathbf{Q}}(0, 2) = 0$. Luego, $\Phi_{\mathbf{Q}}(n) = 0 = Pred(n)$. Ahora bien, si $n > 0$ entonces $Ctrl^2_{\mathbf{Q}}(n, 1) = Ctrl_{\mathbf{Q}}(D_1(n, 2)) = Ctrl_{\mathbf{Q}}(n - 1, 2) = n - 1$. Por lo tanto, $\Phi_{\mathbf{Q}}(n) = n - 1 = Pred(n)$. Concluimos que $\Phi_{\mathbf{Q}} = Pred$, es decir que $Pred$ es computable. □

Ejemplo 1.1.7. Sean $n, i \in \omega$ tal que $1 \leq i \leq n$. La función $\pi^n_i : \omega^n \to \omega$ tal que $\pi^n_i(x_1, ..., x_n) = x_i$ es computable.

Demostración. Sea $\mathbf{P}$ el programa cuya única instrucción es $T_{i,1}$, o bien $r_1 \leftarrow r_i$. Sean $x_1, ..., x_{n-1} \in \omega$, tenemos que $Ctrl^2_{\mathbf{P}}((x_1, ..., x_{n-1}), 1) = Ctrl_{\mathbf{P}}(T_{i,1}((x_1, ..., x_n), 1)) = Ctrl_{\mathbf{P}}((x_i, ..., x_{n-1}), 2) = x_i$. Por lo tanto, para cualesquiera $\Phi^n_{\mathbf{P}}(x_1, ..., x_n) = x_i = \pi^n_i(x_1, ..., x_n)$. Luego $\Phi^n_{\mathbf{P}} = \pi^n_i$, lo cual implica que π^n_i es computable. □

Ejemplo 1.1.8. La función $sg : \omega \to \omega$ tal que

$$sg(x) = \begin{cases} 1 & \text{si} \quad x > 0 \\ 0 & \text{si} \quad x = 0 \end{cases}$$

es computable.

Demostración. Sea $\mathbf{P}$ el siguiente programa:

$\mathbf{I}_1$: **si** $r_1 = r_2$ **entonces**
ir a instrucción número 4
en otro caso
ir a instrucción siguiente
$\mathbf{I}_2 : r_1 \leftarrow 0$
$\mathbf{I}_3 : r_1 \leftarrow r_1 + 1$

Sea $m \in \omega$. Si $m = 0$ entonces $Ctrl^2_{\mathbf{P}}(m, 1) = Ctrl_{\mathbf{P}}(\mathbf{I}_1(m, 1)) = Ctrl_{\mathbf{P}}(m, 4) = m = 0$. Por lo tanto, $\Phi_{\mathbf{P}}(m) = 0 = sg(m)$.
Si $m \geq 1$ entonces

$$\begin{aligned} Ctrl^5_{\mathbf{P}}(m, 1) &= Ctrl^4_{\mathbf{P}}(\mathbf{I}_1(m, 1)) \\ &= Ctrl^3_{\mathbf{P}}(\mathbf{I}_2(m, 2)) \\ &= Ctrl^2_{\mathbf{P}}(\mathbf{I}_3(0, 3)) \\ &= Ctrl_{\mathbf{P}}(1, 4) \\ &= 1 \end{aligned}$$

Por lo tanto, $\Phi_{\mathbf{P}}(m) = 1 = sg(m)$. De esta forma concluimos que $\Phi_{\mathbf{P}} = sg$ y por consiguiente que sg es computable.

□

Ejemplo 1.1.9. Dado $A \subseteq \omega$, la función característica de A, $\chi_A : \omega \to \omega$, es A-computable.

Demostración. Sea $\mathbf{P}^A$ el siguiente programa con oráculo A:

$\mathbf{I}_1$: **si** $r_1 \in A$ **entonces**
$r_1 \leftarrow 1$
en otro caso
$r_1 \leftarrow 0$

Sea $m \in \omega$, entonces $Ctrl^2_{\mathbf{P}^A}(m, 1) = Ctrl_{\mathbf{P}^A}(\mathbf{I}_1(m, 1)) = Ctrl_{\mathbf{P}^A}(\chi_A(m), 2) = \chi_A(m)$. Por lo tanto, $\Phi_{\mathbf{P}^A} = \chi_A$, lo cual nos permite concluir que χ_A es A-computable. □

1.2. Funciones computables

A continuación, nos enfocaremos en buscar herramientas para demostrar con mayor facilidad que una función es computable. Primero, demostraremos que la composición de funciones computables es computable.

Para demostrar esto definiremos, a partir de un programa, un nuevo programa que resultará de hacer una modificación a los índices de las instrucciones del programa dado.

Definición 1.2.1. Dados un número natural $n \in \omega$ y un programa $\mathbf{P}$, posiblemente con oráculo un conjunto A. Definimos la **traslación de P por** n denotado por $\mathbf{P}^{(n)}$ como el programa que se obtiene tras sustituir cada instrucción de $\mathbf{P}$ por la misma instrucción pero con cada uno de sus índices sumados con n.

Por ejemplo si en la definición anterior tenemos que $\mathbf{P}$ tiene a una instrución $r_k \leftarrow r_m$ entonces esta instrucción es sustituida por la instrucción $r_{k+n} \leftarrow r_{m+n}$ en el programa $\mathbf{P}^{(n)}$.

Nos interesa que tras dados dos programas uno pueda ejecutarse si el otro termina. Esto motiva la siguiente definición.

Definición 1.2.2. Sean $\mathbf{P} = (\mathbf{I}_1, ..., \mathbf{I}_k)$ y $\mathbf{Q} = (\mathbf{T}_1, ..., \mathbf{T}_s)$ programas, posiblemente con oráculo. Definimos la **concatenación** de $\mathbf{P}$ con $\mathbf{Q}$, denotado por $\mathbf{PQ}$, como el programa $(\mathbf{L}_1, ..., \mathbf{L}_k, \mathbf{L}_{k+1}, ..., \mathbf{L}_{k+s})$ tal que

1. Para cada $1 \leq i \leq k$, $\mathbf{L}_i = J_{m,n,k+1}$ si $\mathbf{I}_i = J_{m,n,s}$ para algún $s \geq k+1$ y $\mathbf{L}_i = \mathbf{I}_i$ en otro caso.

2. Para cada $1 \leq l \leq s$, $\mathbf{L}_{k+l} = J_{m,n,s+k}$ si $\mathbf{T}_l = J_{m,n,s}$ para algunos $m, n, s \in \omega$ y $\mathbf{L}_{k+l} = T_l$ en otro caso.

Como un programa puede tener instrucciones de almacenamiento y de carga necesitamos añadir una modificación más a los programas que vamos a considerar.

En ocasiones abreviaremos a un conjunto de instrucciones y pensaremos que es una sola instrucción. Por ejemplo dado $m \in \omega$, abreviamos:

$$\text{m-veces} \quad \begin{cases} r_n \leftarrow r_n + 1 \\ \quad \vdots \\ r_n \leftarrow r_n + 1 \end{cases} \qquad \text{Se abrevia por } r_n \leftarrow r_n + m$$

$$\text{m-veces} \quad \begin{cases} r_n \leftarrow r_n \dot{-} 1 \\ \quad \vdots \\ r_n \leftarrow r_n \dot{-} 1 \end{cases} \qquad \text{Se abrevia por } r_n \leftarrow r_n \dot{-} m$$

Ahora bien, cuando una instrucción de almacenamiento o de carga solicite el contenido de un determinado registro para utilizarlo como dirección no tenemos seguro que dicha dirección respeta las traslaciones del programa, es decir pueden existir direcciones de los primeros registros que la traslación no utiliza, lo cual nos lleva a la posibilidad de producir un resultado erróneo. Por ello damos la siguiente definición.

Definición 1.2.3. Sean $m \in \mathbf{P}$ un programa y $\mathbf{P}$ un programa. Sea $\mathbf{P}^{[m]}$ el programa $\mathbf{P}$ con la siguiente modificación: si $\mathbf{P}$ tiene como instrucción a $r_k \leftarrow r_{r_n}$ ó a $r_{r_n} \leftarrow r_k$ entonces agregamos al programa inmediatamente antes de esta instrucción a $r_n \leftarrow r_n + m$ e inmediatamente después las instrucciones $r_n \leftarrow r_n \dot{-} m$.

Dicho lo anterior, ya podemos demostrar nuestro primer resultado relevante.

Proposición 1.2.1. *Sean $A \subseteq \omega$, $k, n \in \mathbb{N}$ y $f : \omega^k \rightharpoonup \omega$, $g_1, ..., g_k : \omega^n \rightharpoonup \omega$ funciones parcialmente A-computables. Entonces la función parcial $h : \omega^n \rightharpoonup \omega$ tal que*

$$h(x_1, ..., x_n) = \begin{cases} f(g_1(x_1, ..., x_n), ..., g_k(x_1, ..., x_n)) & si \ \forall\, 1 \leq i \leq k \ ((x_1, ..., x_n) \in dom\ g_i \\ & \quad \wedge \ g_i(x_1, ..., x_n) \in dom\ f) \\ indefinida & si\ ocurre\ otro\ caso \end{cases}$$

es parcialmente A-computable.

Demostración. Sean $\mathbf{F}, \mathbf{G}_1, ..., \mathbf{G}_k$ los programas que hacen a las funciones $f, g_1, ..., g_k$ parcialmente A-computables respectivamente. Sea $m = n + k$. Definimos como $\mathbf{H}$ al siguiente programa:

$$\left.\begin{array}{l} r_{m+1} \leftarrow r_1 \\ \quad \vdots \\ r_{m+n} \leftarrow r_n \end{array}\right\}$$

Pasamos la entrada $(x_1, ..., x_n)$ a los registros r_{m+1} a r_{m+n}.

$$(\mathbf{G}_1^{[m]})^{(m)}$$

Ejecutamos el programa de la función g_1 en la entrada del programa que ahora se encuentra en los registros r_{m+1} a r_{m+n} y si existe el resultado de este cómputo lo deposita en el registro r_{m+1}.

$$r_{n+1} \leftarrow r_{m+1}$$

El resultado del cómputo anterior, si existe, es el valor de $g_1(x_1, ..., x_n)$ y se guarda en el registro r_{n+1}.

$$\left.\begin{array}{l} r_{m+1} \leftarrow r_1 \\ \quad \vdots \\ r_{m+n} \leftarrow r_n \end{array}\right\}$$

De nuevo pasamos la entrada $(x_1, ..., x_n)$ a los registros r_{m+1} a r_{m+n} ya que en el cómputo anterior se pudieron modificar los contenidos de dichos registros.

$$\left.\begin{array}{l} (\mathbf{G}_2^{[m]})^{(m)} \\ \quad \vdots \\ (\mathbf{G}_k^{[m]})^{(m)} \end{array}\right\}$$

Se repite el proceso con los programas de las demás funciones g_i.

$$r_{n+k} \leftarrow r_{m+1}$$

El resultado del computo anterior, si existe, es el valor de $g_k(x_1, ..., x_n)$ y se guarda al registro r_{n+k}.

$$\left.\begin{array}{l} r_{m+1} \leftarrow r_{n+1} \\ \quad \vdots \\ r_{m+k} \leftarrow r_{n+k} \end{array}\right\}$$

Los resultados guardados en los pasos anteriores $g_1(x_1, ..., x_n), ..., g_k(x_1, ..., x_n)$ se usarán como la entrada del programa de la función f y se colocarán en los registros r_{m+1} a r_{m+k}.

$(\mathbf{F}^{[m]})^{(m)}$	Se ejecuta el programa de la función f en la entrada mencionada y el resultado es el valor $f(g_1(x_1,...,x_n),...,g_k(x_1,...,x_n))$ si existe.
$r_1 \leftarrow r_{m+1}$	Se coloca el resultado obtenido del cómputo anterior en el registro r_1 para devolverlo como resultado final del programa.

Sean $x_1,...,x_n \in \omega$ tales que $(x_1,...,x_n) \in \text{dom } g_i$ y $g_i(x_1,...,x_n) \in \text{dom } f$ para cada $1 \leq i \leq k$. Sean l_i el número de instrucciones de $(\mathbf{G}_i^{[m]})^{(m)}$, t_i+1 los tiempos de ejecución de $(\mathbf{G}_i^{[m]})^{(m)}$ en $(x_1,...,x_n)$, l la cantidad de instrucciones de $(\mathbf{F}^{[m]})^{(m)}$ y $t+1$ el tiempo de ejecución de $\mathbf{F}$ en $(g_1(x_1,...,x_n),...,g_k(x_1,...,x_n))$. Sean $j = \sum_{i=1}^{k}(n+l_i+1)+k+l+1$ y $r = \sum_{i=1}^{k}(n+t_i+1)+k+t+2$. Entonces tenemos que,

$$
\begin{aligned}
&Ctrl^r_{\mathbf{H}}((x_1,...,x_n),1) = Ctrl^{r-1}_{\mathbf{H}}(Ctrl_{\mathbf{H}}((x_1,...,x_n),1))\\
&= Ctrl^{r-1}_{\mathbf{H}}((x_1,...,x_n,0,...,0,\underbrace{x_1}_{\text{posición } m+1}),2)\\
&= \cdots\\
&= Ctrl^{r-n}_{\mathbf{H}}((x_1,...,x_n,0,...,0,\underbrace{x_1}_{m+1},...,\underbrace{x_n}_{m+n}),n+1)\\
&= \cdots\\
&= Ctrl^{r-(n+t_1)}_{\mathbf{H}}((x_1,...,x_n,0,...,0,\underbrace{y_{1_1}}_{m+1},...,y_{1_{a_1}}),n+l_i+1)\\
&= Ctrl^{r-(n+t_1+1)}_{\mathbf{H}}((x_1,...,x_n,y_{1_1},...,0,\underbrace{y_{1_1}}_{m+1},...,y_{1_{a_1}}),n+l_i+2)\\
&= \cdots\\
&= Ctrl^{1+k+t+2}_{\mathbf{H}}((x_1,...,x_n,y_{1_1},...,0,\underbrace{y_{k_1}}_{m+1},...,y_{k_{a_k}}),j-(k+l+1))\\
&= Ctrl^{k+t+2}_{\mathbf{H}}((x_1,...,x_n,y_{1_1},...,y_{k_1},\underbrace{y_{k_1}}_{m+1},...,y_{k_{a_k}}),(j-(k+l+1))+1)\\
&= Ctrl^{(k+t+2)-1}_{\mathbf{H}}((x_1,...,x_n,y_{1_1},...,y_{k_1},\underbrace{y_{1_1}}_{m+1},...,y_{k_{a_k}}),(j-(k+l+1))+2)\\
&= \cdots\\
&= Ctrl^{t+2}_{\mathbf{H}}((x_1,...,x_n,y_{1_1},...,y_{k_1},\underbrace{y_{1_1}}_{m+1},...,y_{k_1},r_1,...,r_q),j-(l+1)+1)\\
&= \cdots\\
&= Ctrl^{2}_{\mathbf{H}}((x_1,...,x_n,y_{1_1},...,y_{k_1},\underbrace{z_1}_{m+1},...,z_s),j)\\
&= Ctrl_{\mathbf{H}}((z_1,...,x_n,y_{1_1},...,y_{k_1},\underbrace{z_1}_{m+1},...,z_s),j+1)\\
&= z_1
\end{aligned}
$$

Como $y_{i_1} = \Phi^n_{(\mathbf{G}_i^{[m]})^{(m)}}(x_1,...,x_n) = g_i(x_1,...,x_n)$ y $z_1 = \Phi^k_{(\mathbf{F}^{[m]})^{(m)}}(y_{1_1},...,y_{k_1}) =$

$f(y_{1_1}, ..., y_{k_1})$, tenemos que $h(x_1, ..., x_n) = z_1$. De esta forma, $\Phi^n_{\mathbf{H}}(x_1, ..., x_n) = h(x_1, ..., x_n)$. En las igualdades anteriores es posible observar que si $(x_1, ..., x_n) \notin \text{dom } g_i$ para alguna i entonces $(x_1, ..., x_n) \notin \text{dom } \Phi^n_{(\mathbf{G}_i^{[m]})^{(m)}}$, por lo que $(\mathbf{G}_i^{[m]})^{(m)}(x_1, ..., x_n) \uparrow$. Por lo tanto, $\mathbf{H}(x_1, ..., x_n) \uparrow$, es decir $\Phi^n_{\mathbf{H}}$ está indefinida en $(x_1, ..., x_n)$. De manera similar ocurre si $(x_1, ..., x_n) \in \text{dom } g_i$ para todo $1 \leq i \leq k$ pero existe $1 \leq i_0 \leq k$ tal que $g_{i_0}(x_1, ..., x_n) \notin \text{dom } f$. Por lo tanto, podemos concluir que $\Phi^n_{\mathbf{H}} = h$, es decir que h es parcialmente A-computable. □

Un ejemplo más de funciones computables es la función que devuelve el primer valor y para el cual una función valuada en y devuelve 0. A esta operación la llamaremos operación de *minimización*.

Proposición 1.2.2. *Sean $A \subseteq \omega$, $n \in \omega$ y $f : \omega^{n+1} \nrightarrow \omega$ una función parcialmente A-computable. La función parcial $h : \omega^n \nrightarrow \omega$ tal que:*

$$h(x_1, ..., x_n) = \begin{cases} \min\limits_{y \in \omega}\{f(x_1, ..., x_n, y) = 0\} & si\ \exists y \in \omega\ (f(x_1, ..., x_n, y) = 0 \\ & \wedge \forall z \leq y((x_1, ..., x_n, z) \in dom\ f)) \\ indefinida & si\ ocurre\ otro\ caso \end{cases}$$

es parcialmente A-computable.

Demostración. Sea $\mathbf{F}$ el programa que hace a f una función parcialmente A-computable. Definimos $m = n + 3$. Denotamos por $\mathbf{H}$ al siguiente programa escrito en pseudocódigo:

$r_{n+1} \leftarrow 0$	Aseguramos que el contenido del registro r_{n+1} es 0. Este registro contendrá los sucesivos valores de y.
$r_{n+2} \leftarrow 0$	Aseguramos que el contenido del registro r_{n+2} es 0. Este registro contendrá un valor 0 que nos nos servirá para compararlo con el valor del registro r_{n+3} y así poder determinar si el registro r_{n+3} tiene el valor 0.
$r_{n+3} \leftarrow r_{n+3} + 1$	Aseguramos que el contenido de r_{n+3} es mayor a cero. En r_{n+3} se guardará el resultado de $f(x_1, ..., x_n, y)$ para alguna tupla $(x_1, ..., x_n, y)$.

hacer	Se inicia un ciclo que se ejecuta al menos una vez y termina cuando se cumple que r_{n+3} es 0. Por la forma en la que utilizaremos a dicho registro, esto ocurre cuando $f(x_1, ..., x_n, y) = 0$ para alguna tupla $(x_1, ..., x_n, y)$.
$r_{m+1} \leftarrow r_1$ $\vdots$ $r_{m+n+1} \leftarrow r_{n+1}$	Se guarda la entrada $(x_1, ..., x_n, y)$ en los registros r_{m+1} a r_{m+n+1}
$(\mathbf{F}^{[m]})^{(m)}$	Se ejecuta el programa que corresponde a f en la entrada $(x_1, ..., x_n, y)$ y devuelve el valor $f(x_1, ..., x_n, y)$ si existe.
$r_{n+3} \leftarrow r_{m+1}$	El resultado del cómputo anterior se coloca en el registro r_{n+3}.
$r_{n+1} \leftarrow r_{n+1} + 1$	Sumamos uno al valor de y.
mientras $r_{n+3} \neq 0$	El ciclo termina si $f(x_1, ..., x_n, y) = 0$, es decir si el registro r_{n+3} tiene el valor 0.
$r_{n+1} \leftarrow r_{n+1} \dot{-} 1$	Se disminuye en uno el valor de y. Y obtenemos el valor y tal que $f(x_1, ..., x_n, y) = 0$.
$r_1 \leftarrow r_{n+1}$	Se prepara el resultado para ser devuelto por el programa.

En el apéndice mostramos los detalles para probar que $\mathbf{H}$ es en efecto un programa RAM.

Sean $x_1, ..., x_n \in \omega$ tales que existe $y \in \omega$ el menor número natural tal que $f(x_1, ..., x_n, y) = 0$ y para todo $j \leq y$, $(x_1, ..., x_n, j) \in \text{dom } f$. Sea l el número de instrucciones de $(\mathbf{F}^{[m]})^{(m)}$ Sean $t_j + 1$ los tiempos de ejecución de $(\mathbf{F}^{[m]})^{(m)}$ en $(x_1, ..., x_n, j)$ para cada $j \leq y$. Sean $j = 3 + 1 + (n + 1) + l + 3 + 2$ y $r = 3 + \sum_{j=0}^{y}(1 + (n + 1) + t_j + 3) + 4$.

$$
\begin{aligned}
&Ctrl^r_{\mathbf{H}}((x_1, ..., x_n), 1) = \ Ctrl^{r-1}_{\mathbf{H}}(Ctrl_{\mathbf{H}}((x_1, ..., x_n), 1)) \\
&= \ Ctrl^{r-1}_{\mathbf{H}}((x_1, ..., x_n, 0), 2) \\
&= \ Ctrl^{r-2}_{\mathbf{H}}((x_1, ..., x_n, 0, 0), 3) \\
&= \ Ctrl^{r-3}_{\mathbf{H}}((x_1, ..., x_n, 0, 0, 1), 4) \\
&= \ Ctrl^{r-4}_{\mathbf{H}}((x_1, ..., x_n, 0, 0, 1), 5) \\
&= \ \cdots \\
&= \ Ctrl^{r-(4+n+1)}_{\mathbf{H}}((x_1, ..., x_n, 0, 0, 1, \underbrace{x_1}_{m+1}, ..., \underbrace{x_n}_{m+n}, 0), 4 + n + 1 + 1) \\
&= \ \cdots \\
&= \ Ctrl^{r-(4+n+1+t_0)}_{\mathbf{H}}((x_1, ..., x_n, 0, 0, 1, \underbrace{z_{0_1}}_{m+1}, ..., z_{0_{s_0}}), 4 + n + 1 + l + 1)
\end{aligned}
$$

$$
\begin{aligned}
&= Ctrl_{\mathbf{H}}^{r-(4+n+1+t_0+1)}((x_1,...,x_n,0,0,z_{0_1},\underbrace{z_{0_1}}_{m+1},...,z_{0_{s_0}}),4+n+1+l+2)\\
&= Ctrl_{\mathbf{H}}^{r-(4+n+1+t_0+2)}((x_1,...,x_n,1,0,z_{0_1},\underbrace{z_{0_1}}_{m+1},...,z_{0_{s_0}}),4+n+1+l+3)\\
&= Ctrl_{\mathbf{H}}^{r-(4+n+1+t_0+3)}((x_1,...,x_n,1,0,z_{0_1},\underbrace{z_{0_1}}_{m+1},...,z_{0_{s_0}}),4)\\
&= \cdots\\
&= Ctrl_{\mathbf{H}}^{4}((x_1,...,x_n,y+1,0,z_{y_1},\underbrace{z_{y_1}}_{m+1},...,z_{y_{s_y}}),4)\\
&= Ctrl_{\mathbf{H}}^{3}((x_1,...,x_n,y+1,0,z_{y_1},\underbrace{z_{y_1}}_{m+1},...,z_{y_{s_y}}),j-1)\\
&= Ctrl_{\mathbf{H}}^{2}((x_1,...,x_n,y,0,z_{y_1},\underbrace{z_{y_1}}_{m+1},...,z_{y_{s_y}}),j)\\
&= Ctrl_{\mathbf{H}}((y,...,x_n,y,0,z_{y_1},\underbrace{z_{y_1}}_{m+1},...,z_{y_{s_y}}),j+1)\\
&= y
\end{aligned}
$$

Por lo tanto, $\Phi_{\mathbf{H}}^n(x_1,...,x_n) = y = h(x_1,...,x_n)$. Ahora bien, dados $x_1,...,x_n \in \omega$ si no existiera $y \in \omega$ tal que $f(x_1,...,x_n,y) = 0$ y $(x_1,...,x_n,z) \in \text{dom } f$ para todo $z \in \omega$ entonces en el desarrollo anterior podemos ver que la instrucción $j-1$ nunca se ejecuta y por lo tanto $\mathbf{H}$ nunca se detiene en la entrada $(x_1,...,x_n)$, es decir $\mathbf{H}(x_1,...,x_n)\uparrow$. Si, por otro lado, existe el menor número natural $y \in \omega$ tal que $f(x_1,...,x_n,y) = 0$ pero existe $z \leq y$ tal que $(x_1,...,x_n,z) \notin \text{dom } f$, entonces el programa $(\mathbf{F}^{[m]})^{(m)}$ nunca se detiene en la tupla $(x_1,...,x_n,0,0,1,x_1,...,x_n,0)$ si $z = 0$ o si $z > 0$ en la tupla $(x_1,...,x_n,z,0,z_{k_1},x_1,...,x_n,z,...,z_{k_{s_k}})$, lo cual implica que $\mathbf{H}(x_1,...,x_n)\uparrow$. Por lo tanto, podemos concluir que $\Phi_{\mathbf{H}}^n = h$, es decir que h es parcialmente A-computable.

□

Una operación muy conocida para obtener funciones es la construcción de éstas por *recursión*. Para coincidir con la literatura denominaremos a esta operación por *recursión primitiva* en lugar de recursión. A continuación mostraremos la existencia de esta operación.

Proposición 1.2.3. *Sean $n \in \omega$, $f : \omega^n \to \omega$ y $g : \omega^{n+2} \to \omega$ funciones. Entonces existe una única función $h : \omega^{n+1} \to \omega$ tal que:*

1. $h(x_1,...,x_n,0) = f(x_1,...,x_n)$
2. $\forall m \in \omega : h(x_1,...,x_n,m+1) = g(x_1,...,x_n,m,h(x_1,...,x_n,m))$

Demostración. Demostremos la existencia de esta función. Por recursión sobre los números naturales, definamos funciones h_n tal que:

1. $h_0 : \omega^n \times \{0\} \to \omega$ tal que:

$$h_0(x_1, ..., x_n, 0) = f(x_1, ..., x_n)$$

2. Suponiendo que hemos definido h_n definimos $h_{n+1} : \omega^n \times \{0, 1, ..., n+1\} \to \omega$ tal que:

$$h_{n+1}(x_1, ..., x_n, k) = \begin{cases} h_n(x_1, ..., x_n, k) & \text{si} \quad 0 \leq k \leq n \\ g(x_1, ..., x_n, n, h_n(x_1, ..., x_n, n)) & \text{si} \quad k = n+1 \end{cases}$$

Notemos que para cada $n \in \omega$, $h_n \subseteq h_{n+1}$. Por lo tanto, $h = \bigcup_{n \in \omega} h_n$ es una función tal que dom $h = \bigcup_{n \in \omega} (\omega^n \times \{0,1,...,n\}) = \omega^n \times \omega = \omega^{n+1}$, cuyo codominio es ω y que por construcción satisface las condiciones requeridas.

Demostremos la unicidad de esta función. Sean $h, h' : \omega^{n+1} \to \omega$ funciones que cumplen lo requerido por el teorema. Demostremos por inducción sobre $m \in \omega$ que $h(x_1, ..., x_n, m) = h'(x_1, ..., x_n, m)$ para cada $x_1, ..., x_n \in \omega$.

$$\begin{aligned} h(x_1, ..., x_n, 0) &= f(x_1, ..., x_n) \\ &= h'(x_1, ..., x_n, 0) \\ h(x_1, ..., x_n, m+1) &= g(x_1, ..., x_n, m, h(x_1, ..., x_n, m)) \\ &= g(x_1, ..., x_n, m, h'(x_1, ..., x_n, m)) \\ &= h'(x_1, ..., x_n, m+1) \end{aligned}$$

Concluimos que $h = h'$ y por consiguiente que la función h es única. □

La proposición anterior nos permite establecer la siguiente definición.

Definición 1.2.4. Sea $n \in \omega$. La operación de **recursión primitiva** asocia a las funciones $f : \omega^n \to \omega$ y $g : \omega^{n+2} \to \omega$, la función $h : \omega^{n+1} \to \omega$ que satisface:

1. $h(x_1, ..., x_n, 0) = f(x_1, ..., x_n)$

2. $\forall m \in \omega : h(x_1, ..., x_n, m+1) = g(x_1, ..., x_n, m, h(x_1, ..., x_n, m))$

Veamos que para cualquier $A \subseteq \omega$, la operación de recursión primitiva produce funciones A-computables siempre y cuando se utilicen funciones A-computables en la construcción.

Proposición 1.2.4. *Sean $A \subseteq \omega$, $n \in \omega$, $f : \omega^n \to \omega$ y $g : \omega^{n+2} \to \omega$ funciones A-computables. Si $h : \omega^{n+1} \to \omega$ la función obtenida por recursión primitiva aplicada a las funciones f y g entonces h es A-computable.*

Demostración. Construiremos un programa **H** que cómputa a la función h en base a los programas de las funciones f y g.

Para ello la idea del programa **H** es que si tiene una tupla de la forma $(x_1, ..., x_n, q)$ como entrada entonces calcula sucesivamente los valores de la función h en tuplas de la forma $(x_1, ..., x_n, i)$ para $0 \leq i \leq q$, o bien para valores de la forma $h(i)$ con $0 \leq i \leq q$ cuando $n = 0$.

Para ser más breves en esta demostración haremos el caso $n > 0$, el caso $n = 0$ es similar.

Sean **F** y **G** los programas de las funciones f y g respectivamente. Definimos como $m = n + 3$. Sea **H** el siguiente programa escrito en pseudocódigo que en el apéndice demostramos que es en efecto un programa:

$r_{n+2} \leftarrow 0$	Aseguramos que el contenido del registro r_{n+2} es 0 el cual nos servirá de contador i.
$\left.\begin{array}{l} r_{m+1} \leftarrow r_1 \\ \quad\vdots \\ r_{m+n} \leftarrow r_n \end{array}\right\}$	Pasamos la entrada $(x_1, ..., x_n)$ a los registros r_{m+1} a r_{m+n}.
$(\mathbf{F}^{[m]})^{(m)}$	Ejecutamos el programa de la función f en la entrada inicial con lo cual obtenemos $f(x_1, ..., x_n)$.
$r_{n+3} \leftarrow r_{m+1}$	Guardamos el resultado del cómputo anterior en el registro r_{n+3}.
mientras $r_{n+1} \neq r_{n+2}$	Inicializamos un ciclo en el que obtendremos de manera sucesiva los valores de la forma $g(x_1, ..., x_n, i, h(x_1, ..., x_n, i))$. Este ciclo se detendrá cuando nuestro contador en el registro r_{n+2} alcance el valor de q, el cual se encuentra inicialmente en el registro r_{n+1}.
$\left.\begin{array}{l} r_{m+1} \leftarrow r_1 \\ \quad\vdots \\ r_{m+n} \leftarrow r_n \end{array}\right\}$	Pasamos la entrada $(x_1, ..., x_n)$ a los registros r_{m+1} a r_{m+n} ya que en cómputos anteriores dichos registros pudieron cambiar.
$r_{m+n+1} \leftarrow r_{n+2}$	Mandamos el valor i que se encuentra en el registro r_{n+2} al registro r_{m+n+1} para que realicemos una ejecución del programa de g.
$r_{m+n+2} \leftarrow r_{n+3}$	Mandamos a r_{m+n+2} el valor del resultado de cómputos anteriores de tipo $g(x_1, ..., x_n, i, h(x_1, ..., x_n, i))$ o bien de tipo $f(x_1, ..., x_n)$.

$(\mathbf{G}^{[m]})^{(m)}$	Ejecutamos el programa de g y obtenemos el valor de $g(x_1, ..., x_n, i, h(x_1, ..., x_n, i))$ para algún $i \in \omega$.
$r_{n+3} \leftarrow r_{m+1}$	Guardamos el resultado obtenido anteriormente si existe para una nueva posible iteración.
$r_{n+2} \leftarrow r_{n+2} + 1$	Aumentamos en uno el contador i.
fin mientras	Finaliza el ciclo si se cumple la condición de que el contador i es igual a q.
$r_1 \leftarrow r_{n+3}$	Se coloca el resultado final para que sea devuelto por el programa.

Sean $l_1, l_2 \in \mathbb{N}$ el número de instrucciones de $(\mathbf{F}^{[m]})^{(m)}$ y $(\mathbf{G}^{[m]})^{(m)}$ respectivamente. Sea $t+1$ el tiempo de ejecución de $(\mathbf{F}^{[m]})^{(m)}$ en $(x_1, ..., x_n)$, si está definido. Dado $i \in \omega$, sea $t_i + 1$ el tiempo de ejecución de $(\mathbf{G}^{[m]})^{(m)}$ en $(x_1, ..., x_n, i, h(x_1, ..., x_n, i))$.

Sean $x_1, ..., x_n, q \in \omega$. Sea $j = 1 + n + l_1 + 2 + n + l_2 + 3 + 1$. Supongamos que $q = 0$. Definimos $r = 1 + n + t + 4$. Entonces tenemos que:

$$
\begin{aligned}
& Ctrl^r_{\mathbf{H}}((x_1, ..., x_n, q), 1) = \ Ctrl^{r-1}_{\mathbf{H}}(Ctrl_{\mathbf{H}}((x_1, ..., x_n, q), 1)) \\
&= \ Ctrl^{r-1}_{\mathbf{H}}((x_1, ..., x_n, q, 0), 2) \\
&= \ Ctrl^{r-2}_{\mathbf{H}}((x_1, ..., x_n, q, 0, 0, \underbrace{x_1}_{m+1}), 3) \\
&= \ \cdots \\
&= \ Ctrl^{r-(1+n)}_{\mathbf{H}}((x_1, ..., x_n, q, 0, 0, \underbrace{x_1}_{m+1}, ..., \underbrace{x_n}_{m+n}), 1 + n + 1) \\
&= \ \cdots \\
&= \ Ctrl^4_{\mathbf{H}}((x_1, ..., x_n, q, 0, 0, \underbrace{z_1}_{m+1}, ..., z_s), 1 + n + l_1 + 1) \\
&= \ Ctrl^3_{\mathbf{H}}((x_1, ..., x_n, q, 0, z_1, \underbrace{z_1}_{m+1}, ..., z_s), 1 + n + l_1 + 2) \\
&= \ Ctrl^2_{\mathbf{H}}((x_1, ..., x_n, q, 0, z_1, \underbrace{z_1}_{m+1}, ..., z_s), j) \\
&= \ Ctrl_{\mathbf{H}}((z_1, ..., x_n, q, 0, z_1, \underbrace{z_1}_{m+1}, ..., z_s), j + 1) \\
&= \ z_1
\end{aligned}
$$

Por lo tanto $\Phi^{n+1}_{\mathbf{H}}(x_1, ..., x_n, q) = \Phi_{(\mathbf{F}^{[m]})^{(m)}}(x_1, ..., x_n) = f(x_1, ..., x_n) = h(x_1, ..., x_n, q)$.

Ahora bien supongamos que $q \geq 1$. Luego existe $q' \in \omega$ tal que $q = q' + 1$. Sea $r = 1 + n + t + 1 + \sum_{i=0}^{q-1}(1 + n + 2 + t_i + 3) + 3$. Para abreviar sean $a = 1 + n + t + 1$, $b = 1 + n + l_1 + 3$. Por lo tanto,

$$
\begin{aligned}
&Ctrl^r_{\mathbf{H}}((x_1,...,x_n,q),1) = \ Ctrl^{r-1}_{\mathbf{H}}(Ctrl_{\mathbf{H}}((x_1,...,x_n,q),1))\\
&= \ Ctrl^{r-1}_{\mathbf{H}}((x_1,...,x_n,q,0),2)\\
&= \ Ctrl^{r-2}_{\mathbf{H}}((x_1,...,x_n,q,0,0,\underbrace{x_1}_{m+1}),3)\\
&= \ \cdots\\
&= \ Ctrl^{r-(1+n)}_{\mathbf{H}}((x_1,...,x_n,q,0,0,\underbrace{x_1}_{m+1},...,x_n),1+n+1)\\
&= \ \cdots\\
&= \ Ctrl^{r-(1+n+t)}_{\mathbf{H}}((x_1,...,x_n,q,0,0,\underbrace{z_1}_{m+1},...,z_s),1+n+l_1+1)\\
&= \ Ctrl^{r-(1+n+t+1)}_{\mathbf{H}}((x_1,...,x_n,q,0,z_1,\underbrace{z_1}_{m+1},...,z_s),1+n+l_1+2)\\
&= \ Ctrl^{r-(1+n+t+1+1)}_{\mathbf{H}}((x_1,...,x_n,q,0,z_1,\underbrace{z_1}_{m+1},...,z_s),1+n+l_1+3)\\
&= \ Ctrl^{r-(a+1)}_{\mathbf{H}}((x_1,...,x_n,q,0,z_1,\underbrace{x_1}_{m+1},...,z_s),1+n+l_1+4)\\
&= \ \cdots\\
&= \ Ctrl^{r-(a+1+n)}_{\mathbf{H}}((x_1,...,x_n,q,0,z_1,\underbrace{x_1}_{m+1},...,x_n,...,z_s),b+n+1)\\
&= \ Ctrl^{r-(a+1+n+1)}_{\mathbf{H}}((x_1,...,x_n,q,0,z_1,\underbrace{x_1}_{m+1},...,x_n,0,...,z_s),b+n+2)\\
&= \ Ctrl^{r-(a+1+n+2)}_{\mathbf{H}}((x_1,...,x_n,q,0,z_1,\underbrace{x_1}_{m+1},...,x_n,0,z_1,...,z_s),b+n+3)\\
&= \ \cdots\\
&= \ Ctrl^{r-(a+1+n+2+t_0)}_{\mathbf{H}}((x_1,...,x_n,q,0,z_1,\underbrace{z_{1_1}}_{m+1},...,z_{1_{s_1}},...,u_{s_1}),b+n+2+l_2+1)\\
&= \\
&\ Ctrl^{r-(a+1+n+2+t_0+1)}_{\mathbf{H}}((x_1,...,x_n,q,0,z_{1_1},\underbrace{z_{1_1}}_{m+1},...,z_{1_{s_1}},...,u_{s_1}),b+n+2+l_2+2)\\
&= \\
&\ Ctrl^{r-(a+1+n+2+t_0+2)}_{\mathbf{H}}((x_1,...,x_n,q,1,z_{1_1},\underbrace{z_{1_1}}_{m+1},...,z_{1_{s_1}},...,u_{s_1}),b+n+2+l_2+3)\\
&= \ Ctrl^{r-(a+1+n+2+t_0+3)}_{\mathbf{H}}((x_1,...,x_n,q,1,z_{1_1},\underbrace{z_{1_1}}_{m+1},...,z_{1_{s_1}},...,u_{s_1}),b+n+2)\\
&= \ \cdots\\
&= \ Ctrl^{3}_{\mathbf{H}}((x_1,...,x_n,q,q,z_{q_1},\underbrace{z_{q_1}}_{m+1},...,z_{q_{s_q}},...,u_{s_q}),b+n+2)\\
&= \ Ctrl^{2}_{\mathbf{H}}((x_1,...,x_n,q,q,z_{q_1},\underbrace{z_{q_1}}_{m+1},...,z_{q_{s_q}},...,u_{s_q}),j)\\
&= \ Ctrl_{\mathbf{H}}((z_{q_1},...,x_n,q,q,z_{q_1},\underbrace{z_{q_1}}_{m+1},...,z_{q_{s_q}},...,u_{s_q}),j+1)\\
&= \ z_{q_1}
\end{aligned}
$$

Notemos que $\Phi_{\mathbf{H}}^{n+1}(x_1, ..., x_n, q) = z_{q_1} = g(x_1, ..., x_n, q', h(x_1, ..., x_n, q')) = h(x_1, ..., x_n, q)$. Por lo tanto, $\Phi_{\mathbf{H}}^{n+1} = h$, lo cual implica que h es parcialmente A-computable.

□

A continuación mostraremos ejemplos de funciones computables, únicamente definiéndolas por recursión primitiva.

Ejemplo 1.2.1. En la siguiente relación, en la columna central mostramos formalmente que las funciones en la columna izquierda son primitivas recursivas partiendo de funciones que ya sabemos que son computable y en la columna derecha la forma usual de hacer de demostrarlo, que es sin escribir las funciones a las cuales se les aplica la operación. Abreviamos por *R.P.* a Recursión Primitiva.

Función	**Definición Formal por R.P.**	**Definición Informal**
$(x, y) \to x + y$	$f(x) = \pi_1(x) = x$; $g(x, y, z) =$ $\pi_3(x, y, z) + 1 = z + 1$	$x + 0 = x$ $x + (y + 1) = (x + y) + 1$
$(x, y) \to x \cdot y$	$f(x) = C(x) = 0$; $g(x, y, z) =$ $\pi_3(x, y, z) + 1 = z + 1$	$x \cdot 0 = 0$ $x \cdot (y + 1) = (x \cdot y) + 1$
$x \dot{-} y = \begin{cases} x - y & \text{si} \quad x \geq y \\ 0 & \text{si} \quad x < y \end{cases}$	$f(x) = \pi_1(x) = x$; $g(x, y, z) =$ $\text{Pred}(\pi_3(x, y, z)) =$ $\text{Pred}(z)$	$x \dot{-} 0 = 0$ $x \dot{-} (y + 1) = (x \dot{-} y) \dot{-} 1$
$(x, y) \to x^y$ si $x > 0$ e indefinida en otro caso	$f(x) = C(x) + 1 = 1$; $g(x, y, z) = \pi_3(x, y, z) \cdot$ $\pi_1(x, y, z) = z \cdot x$	$x^0 = 1$ $x^{y+1} = x^y \cdot x$
$x \to x!$	$f = 1$ (f es la constante 1); $g(x, y) = x \cdot y$	$0! = 1$ $(x + 1)! = (x + 1)x!$

Una definición que será muy importante en lo sucesivo es la siguiente:

Definición 1.2.5. Sean $A, B \subseteq \omega$.

(i) Decimos que A es **B-computable** si y sólo la función característica de A, χ_A, es B-computable.

(ii) Decimos que A **es computable** si y sólo si es A es $\emptyset$-computable.

Definición 1.2.6. Sean $A \subseteq \omega$, $n \in \mathbb{N}$ y P un predicado de aridad n. Entonces,

(i) Decimos que P **es A-computable** si y sólo si el conjunto $\{(x_1, ..., x_n) \in \omega^n : P(x_1, ..., x_n)\}$ es A-computable.

(ii) P **es computable** si y sólo si P es $\emptyset$-computable.

En el siguiente lema, mostramos más ejemplos de funciones computables combinando también la construcción de funciones por minimización.

Lema 1.2.1. *(i) Las siguientes funciones son computables:*

(a) $(x, y) \to |x - y|$
(b) $(x, y) \to \text{mín}\{x, y\}$
(c) $(x, y) \to \text{máx}\{x, y\}$

(ii) Sean P y Q predicados computables de la misma aridad, entonces los siguientes predicados son computables:

(a) $\neg P$

(b) $P \wedge Q$

(c) $P \vee Q$

(iii) Si $f : \omega \times \omega \to \omega$ es una función computable entonces las funciones $f_1, f_2 : \omega \times \omega \to \omega$ tal que

(a) $f_1(x, y) = \sum_{z \leq y} f(x, z)$
(b) $f_2(x, y) = \prod_{z \leq y} f(x, z)$

son computables.

(iv) (a) Si $g : \omega \times \omega \to \omega$ es una función computable entonces la función $M_g : \omega \times \omega \to \omega$ tal que

$$M_g(x, y) = \begin{cases} \text{mín}_{z \leq y}\{g(x, z) = 0\} & \text{si} \quad \text{existe } z_0 \leq y \text{ tal que } f(x, z_0) = 0 \\ y + 1 & \text{en cualquier otro caso} \end{cases}$$

es computable.

(b) Si $R(x,y)$ es un predicado computable, entonces la función $M_R : \omega \times \omega \to \omega$ tal que

$$M_R(x,y) = \begin{cases} \min\limits_{z \leq y}\{R(x,z)\} & si \quad existe\ z_0 \leq y\ tal\ que\ R(x,z_0) \\ y+1 & en\ cualquier\ otro\ caso \end{cases}$$

es computable.

(v) Las siguientes funciones son parcialmente computables, excepto div y D que son computables.

(a) $qt(x,y) = \begin{cases} cociente\ de\ dividir\ y\ entre\ x & si \quad x \neq 0 \\ indefinido & si \quad x = 0 \end{cases}$

(b) $res(x,y) = \begin{cases} residuo\ tras\ dividir\ y\ entre\ x & si \quad x \neq 0 \\ indefinido & si \quad x = 0 \end{cases}$

(c) $div(x,y) = \begin{cases} 1 & si \quad x\ divide\ a\ y \\ 0 & si \quad x\ no\ divide\ a\ y \end{cases}$

(d) $D(x) =$ *número de divisores de* x

(vi) Las siguientes funciones son computables:

(a) $Pr(x) = \begin{cases} 1 & si \quad x\ es\ un\ número\ primo \\ 0 & en\ otro\ caso \end{cases}$

(b) $p(x) = \begin{cases} el\ x\text{-}ésimo\ número\ primo & si \quad x \neq 0 \\ 1 & si \quad x = 0 \end{cases}$

(c) $exp(x,y) = \begin{cases} el\ exponente\ del\ y\text{-}ésimo\ número\ primo\ en\ la\ factorización\ por\ números\ primos\ de\ x & si \quad x > 1 \ \wedge \ y > 0 \\ 0 & en\ otro\ caso \end{cases}$

Demostración. *(i)* *(a)* Para cada $x, y \in \omega$, $|x-y| = (x \dot{-} y) + (y \dot{-} x)$. Por lo tanto, $|\cdot|$ es una función computable.

(b) Para cada $x, y \in \omega$, $\max\{x,y\} = x + (y \dot{-} x)$. Por lo tanto, máx es una función computable.

(c) Para cada $x, y \in \omega$, $\min\{x,y\} = x \dot{-} (x \dot{-} y)$. Por lo tanto, mín es una función computable.

(ii) Sean χ_P y χ_Q las funciones características asociadas a los predicados P y Q respectivamente.

(a) Como $\chi_{\neg P} = 1 \dot{-} \chi_P$, tenemos que $\neg P$ es computable.

(b) $\chi_{P\wedge Q} = \text{máx}\{\chi_P, \chi_Q\}$, por lo tanto $P \wedge Q$ es computable.

(c) $\chi_{P\vee Q} = \text{mín}\{\chi_P, \chi_Q\}$, por lo tanto $P \vee Q$ es computable.

(iii) *(a)* La función f_1 se puede definir por recursión primitiva como sigue:

$$f_1(x,0) = f(x,0)$$
$$f_1(x,y+1) = \sum_{z\leq y} f(x,z) + f(x,y+1)$$

Por lo tanto, si f es computable, f_1 también es computable.

(b) De forma similar, la función f_2 se puede definir por recursión primitiva como:

$$f_2(x,0) = f(x,0)$$
$$f_2(x,y+1) = (\prod_{z\leq y} f(x,z)) \cdot f(x,y+1)$$

Por lo tanto, f_2 es computable.

(iv) *(a)* Sea $h : \omega \times \omega \to \omega$ la función definida como $h(x,v) = \prod_{z\leq v} sg(g(x,z))$. Por el inciso *(iii)(b)*, h es una función computable. Ahora bien, si dados $x, y \in \omega$ existe $z_0 \leq y$ tal que $g(x,z_0) = 0$ entonces para cada $v < z_0$ se tiene que $g(x,v') \neq 0$ para cada $v' \leq v$ luego $sg(g(x,v')) = 1$ y por lo tanto $h(x,v) = 1$. Ahora bien, para cada $z_0 \leq v \leq y$, como $g(x,z_0) = 0$, $sg(g(x,z_0)) = 0$ y por lo tanto, $h(x,v) = 0$. De modo que sólo hay z_0 números $v \leq y$ tales que $h(x,v) = 1$. Por lo tanto, $\sum_{v\leq y} h(x,v) = z_0$. Ahora bien, si no existe tal número z_0 entonces $h(x,v) = 1$ para todo $v \leq y$. Por lo tanto, $\sum_{v\leq y} h(x,v) = y+1$. Por último, notemos que $\sum_{v\leq y} h(x,v)$ es una función computable y que $M_g(x,y) = \sum_{v\leq y} h(x,v)$. Por lo tanto, M_g es una función computable.

(b) Sea f la función característica de $\{(x,y) \in \omega \times \omega : R(x,y)\}$. Luego, la función $g(x,y) = 1\dot{-}f(x,y)$ es computable. Notemos que $g(x,y) = 0$ si y sólo si $R(x,y)$. Por lo tanto, dados $x, y \in \omega$ si existe $z_0 \leq y$ tal que $g(x,z_0) = 0$, es decir, $R(x,z_0)$ entonces $M_g(x,y) = z_0 = \min_{z\leq y}\{g(x,z) = 0\} = \min_{z\leq y}\{R(x,z)\} = M_R(x,y)$. En caso contrario, $M_g(x,y) = y+1 = M_R(x,y)$. Por lo tanto, $M_R = M_g$, lo cual implica que M_R es computable.

(v) *(a)* Sea $f : \omega^3 \to \omega$ tal que $f(x,y,z) = x(z+1)\dot{-}y$. Notemos que f es computable. Sea $g : \omega^3 \to \omega$ tal que $g(x,y,z) := 1\dot{-}sg(f(x,y,z))$. Notemos que g es computable. También observemos que para cada $x, y \in \omega$ con $x \neq 0$, existe $z \in \omega$ tal que $y < (z+1)x$ (basta tomar $z = y$), que es equivalente a $0 < (z+1)x\dot{-}y$ y por lo tanto también a $(z+1)x\dot{-}y \neq 0$ y a $g(x,y,z) = 0$. De esta forma, la función parcial

$h : \omega^2 \rightharpoonup \omega$ tal que

$$h(x,y) = \begin{cases} \min\limits_{z\in\omega}\{g(x,y,z) = 0\} & \text{si } x \neq 0 \\ indefinida & \text{si ocurre otro caso} \end{cases}$$

es parcialmente computable (es una función construida por minimización).
Ahora bien, por el algoritmo de la división, dados $x, y \in \omega$ con $x \neq 0$ existen $q, r \in \omega$ tal que $y = qx + r$ con $0 \leq r < x$. De modo que $q = qt(x,y)$. Afirmamos que $h(x,y) = q$. Para ello sea $z = h(x,y)$ y veamos que $z = q$. Luego, $zx \leq y$ por la definición de z. De modo que existe $r \in \omega$ tal que $y = zx + r$. Si $r > x$ entonces $y = zx + r > zx + x = (z+1)x$, lo cual es una contradicción ya que $y < (z+1)x$. Por lo tanto, $0 \leq r < x$. Por la unicidad de z y r tenemos que $z = q$. Por consiguiente, $qt(x,y) = h(x,y)$ para cada $x, y \in \omega$ con $x \neq 0$. De modo que qt es parcialmente computable.

(b) Dados $x, y \in \omega$ con $x \neq 0$, es evidente que $y = qt(x,y)x + res(x,y)$, por lo que $res(x,y) = y \dot{-} qt(x,y)x$. Por lo tanto, res es parcialmente computable.

(c) Sea $f : \omega \times \omega \rightharpoonup \omega$ tal que

$$f(x,y) = \begin{cases} 1 \dot{-} sg(res(x,y)) & \text{si } x \neq 0 \\ indefinida & \text{si ocurre otro caso} \end{cases}$$

Tenemos que f es parcialmente computable. Más áun es evidente que dom $f = \omega \setminus \{0\}$. Ahora bien, dados $x, y \in \omega$ con $x \neq 0$, notemos que x divide a y si y sólo si $res(x,y) = 0$ lo cual ocurre si y sólo si $sg(res(x,y)) = 0$. Por lo tanto, $div(x,y) = 1 = 1 \dot{-} sg(res(x,y)) = f(x,y)$ cuando $x \neq 0$. Sea $\mathbf{F}$ el programa, digamos con k instrucciones, que hace a f parcialmente computable. Definimos $\mathbf{D}$ como el programa:

si $r_1 = r_3$ **entonces**
    ir a instrucción número $k+2$
en otro caso
    ir a instrucción siguiente
F

Por consiguiente sean $x, y \in \omega$, si $x \neq 0$, $\Phi^2_{\mathbf{H}}(x,y) = \Phi^2_{\mathbf{F}}(x,y) = f(x,y) = div(x,y)$. Si $x = 0$ entonces $\Phi^2_{\mathbf{H}}(x,y) = x = 0 = div(x,y)$. De este modo, $\Phi^2_{\mathbf{H}} = div$ y por lo tanto div es computable.

(d) Sea $y \in \omega$. Un divisor x de y cumple que $div(x,y) = 1$ con $x \leq y$. Por lo tanto, $D(y) = \sum\limits_{x\leq y} div(x,y)$ (la función va sumando uno por cada divisor de y que encuentra). De ahí que D es una función computable.

(vi) *(a)* Un número $x \in \omega$ es un número primo si y sólo si $x > 1$ y $D(x) = 2$, lo cual ocurre si y sólo si $D(x) = 2$ (ya que $D(1) = 1$ y $D(0) = 0$). Por lo tanto, $Pr(x) = sg(x)[1 \dot{-} sg(|D(x) - 2|)]$. De ahí que Pr es computable.

(b) Sea $PR(x)$ el predicado que es verdadero si y sólo si x es un número primo y falso en otro caso. Notemos que P es computable ya que su función característica asociada es Pr, la cual es computable. Observemos también que el predicado $x < y$ es computable ya que $x < y$ si y sólo si $y \dot{-} x \neq 0$, lo cual ocurre si y sólo si $sg(y \dot{-} x) = 1$. Por lo tanto, $sg(y \dot{-} x)$ es la función característica asociada a $x < y$, la cual es computable.
Denotemos por $R(x, z)$ al predicado $PR(z) \wedge (x < z)$. Por lo mencionado anteriormente y el inciso *(ii)(b)* de este lema, $R(x, z)$ es computable.

Sea M_R la función inducida por R del inciso *(iv)(b)*, luego h es una función computable. Y definamos la función $t : \omega \to \omega$ tal que $t(x) = x! + 1$, la cual es computable. Por lo tanto, la función $g(x, y) = M_R(y, t(y))$ es una función computable.

Sea $h : \omega \to \omega$ la función obtenida por recursión primitiva a partir de la constante 1 y la función g. Afirmamos que es $h(x) = p(x)$ para todo $x \in \omega$. Por inducción, $h(0) = 1 = p(0)$. Ahora supongamos que $h(n) = p(n)$ y veamos que $h(n+1) = p(n+1)$. En efecto, como existe al menos un número primo p tal que $p(n) < p \leq p(n)! + 1$ entonces existe $z_0 \leq p(n)! + 1$ tal que $R(p(n), z_0)$. Por lo tanto,

$$\begin{aligned} h(n+1) &= g(n, h(n)) \\ &= M_R(h(n), t(h(n))) \\ &= M_R(p(n), t(p(n))) \\ &= \min_{z \leq p(n)!+1} \{R(p(n), z)\} \\ &= \min_{z \leq p(n)!+1} \{PR(z) \wedge (p(n) < z)\} \end{aligned}$$

Como se calcula el mínimo número $z \in \omega$ tal que $R(p(n), z)$ entonces se devuelve el número primo inmediatamente mayor a $p(n)$, es decir devuelve $p(n+1)$. Por lo tanto, $h(n+1) = p(n+1)$. De este modo concluimos que, $h(x) = p(x)$ para todo $x \in \omega$ y por consiguiente que $h = p$, lo cual implica que p es computable.

(c) Denotemos por $DIV(x, y)$ al predicado que es verdadero si y sólo si $div(x, y) = 1$, es decir, si y sólo si x divide a y, y falso si y sólo si $div(x, y) = 0$, es decir, x no divide a y. Por la definición de DIV, podemos ver que es computable (ya que div es una función computable y es la función característica asociada a DIV). Por otro lado, notemos que la función $t : \omega \times \omega \to \omega$ tal que $t(x, y) = p(x)^{y+1}$ es una función computable. Definamos como $R(x, y, z)$ al predicado $DIV(t(y, z), x)$. La función característica asociada a R es la función $div(t(y, z), x)$, la cual es computable. Por lo tanto, R es un predicado computable. Por un

inciso anterior en este lema, tenemos que $\neg R$ es un predicado computable. Observemos que si $x, y \in \omega$ tal que $x > 1$ y $y > 0$ entonces existe $z \leq x$ tal que $\neg R(x, y, z)$ ya que al menos $\neg R(x, y, x)$. Entonces la función, $k(x, y) = sg(y)sg(x \dot{-} 1) \min\limits_{z \leq x}\{\neg R(x, y, z)\}$ es una función computable ya que basta dar un programa que calcule $sg(y)sg(x \dot{-} 1)$ y de manera sucesiva el menor valor $z \leq x$ tal que $\neg R(x, y, z)$. Notemos por último que $exp(x, y) = k(x, y)$. Por lo tanto, $exp = k$, lo cual implica que exp es una función computable.

□

Habiendo explorado algunas técnicas que nos facilitan la demostración de que una función es computable y mostrado ejemplos, pasamos ahora a un tema importante que nos servirá más adelante.

1.3. Enumeración de programas

En esta sección nos enfocaremos en asociar un número natural de forma computable y biyectiva a una tupla finita de números naturales. Nuestro primer propósito es que con esto podamos asociar un número natural a un programa. También nos será de importancia asociar un número natural al contenido útil de la cinta de una máquina de acceso aleatorio. Por ello damos las siguientes definiciones.

Definición 1.3.1. Definimos **la función de configuración** $c : \omega^{<\mathbb{N}} \to \omega$ tal que para cada $(x_1, ..., x_n) \in \omega^{<\mathbb{N}}$, $c(x_1, ..., x_n) = \prod\limits_{i=1}^{n} p(i)^{x_i+1}$. Diremos que un número natural $m \in \omega$ es una **configuración** si y sólo si $m \in$ rango c.

Observemos que $\omega \neq$ rango c ya que el número $2 \cdot 5 \notin$ rango c. Por lo tanto concluimos que c no es una función sobreyectiva. Pero observemos que c es inyectiva ya que por el Teorema Fundamental de la Aritmética dos números naturales distintos deben tener diferente descomposición por factores primos.

Recordemos que no contamos con una definición para decir que una función de ω^n a ω ó de $\omega^{<\mathbb{N}}$ a ω es «computable». La idea para dar una definición en este sentido es que exista un programa tal que dada una tupla de números naturales como entrada, éste pueda devolver un número natural. Y que además la «función inducida» por este programa coincida con la función que buscamos que sea «computable».

Las siguientes definiciones dan una noción de computabilidad para este tipo de funciones.

Definición 1.3.2. *(i)* Dados $n \in \omega$ con $n > 1$, una función $f : \omega \to \omega^n$ es **n-calculable** si y sólo si existe un programa $\mathbf{P}$, digamos de k instrucciones,

tal que para todo $x \in \omega$ existe $t \in \omega$ tal que si $f(x) = (x_1, ..., x_n)$ entonces $Ctrl^t_{\mathbf{P}}(x, 1) = ((x_1, ..., x_n, 0, ..., \underbrace{0}_{s}), k + 1)$ para algún $s \in \omega$.

(ii) Una función f es **calculable** si y sólo si ocurre alguno de los siguientes casos:

(a) Si $f : \omega \to \omega^{<\mathbb{N}}$ es una función para la cual existe un programa $\mathbf{P}$, digamos de k instrucciones, tal que existe $m \in \mathbb{N}$ tal que para todo $x \in \omega$ existe $t \in \omega$ tal que si $f(x) = (x_1, ..., x_n)$ entonces $Ctrl^t_{\mathbf{P}}(x, 1) = ((0, 0, ..., 0, \underbrace{n}_{\text{posición } m}, x_1, ..., x_n), k + 1)$.

(b) Si $f : \omega^{<\mathbb{N}} \to \omega$ es una función para la cual existe un programa $\mathbf{P}$, digamos de k instrucciones, tal que existe $m \in \mathbb{N}$ tal que para cada $(x_1, ..., x_n) \in \omega^{<\mathbb{N}}$ existen $t \in \omega$ tal que si $f(x_1, ..., x_n) = y_1$ entonces $Ctrl^{t+1}_{\mathbf{P}}((0, .., 0, \underbrace{n}_{\text{posición } m}, x_1, ..., x_n), 1) = y_1$.

La razón por la que queremos devolver también la longitud de una tupla finita es por fines prácticos. De este modo nuestros programas serán más sencillos de describir e implementar.

A continuación mostramos un ejemplo de función calculable.

Proposición 1.3.1. *La función de configuración c es calculable.*

Demostración. La idea de la demostración es que a partir de la longitud de la tupla dada, podemos obtener de forma computable la cantidad de números primos necesarios para calcular la expresión final. Posteriormente obtenemos los valores de los números primos con potencias los elementos de la tupla dada sumados con uno. Finalmente multiplicamos dos a dos los resultados que obtengamos del paso anterior de forma que obtengamos el producto de todos ellos. Estos dos últimos pasos los podemos realizar de forma computable. Para ver más detalles de esta demostración vaya al apéndice.

□

Para poder cumplir con nuestro objetivo de asignar de forma computable números naturales a programas, probaremos que existe una asignación biyectiva y computable entre tuplas finitas de números naturales y números naturales.

Proposición 1.3.2. *(i) La función* $\boldsymbol{\varphi_2} : \omega \times \omega \to \omega$ *tal que*

$$\boldsymbol{\varphi_2}(x, y) = \frac{(x + y + 1)(x + y)}{2} + x$$

es biyectiva, computable y su función inversa $\boldsymbol{\varphi_2^{-1}}$ *es 2-calculable.*

(ii) Dado $n \in \omega$ *con* $n \geq 2$*, existe una función* $\boldsymbol{\varphi_n} : \omega^n \to \omega$ *biyectiva, computable y su función inversa es n-calculable.*

(iii) Existe una función $\boldsymbol{\psi} : \omega^{<\mathbb{N}} \to \omega$ *biyectiva, calculable y su función inversa también es calculable.*

Demostración. *(i)* Es evidente que $\boldsymbol{\varphi_2}$ es computable. Afirmamos que $\boldsymbol{\varphi_2}$ es biyectiva.

Veamos primero que $\boldsymbol{\varphi_2}$ es sobreyectiva. Sea $z \in \omega$. Si $z = 0$, observe que $\boldsymbol{\varphi_2}(0,0) = 0 = z$. Supongamos que $z \geq 1$, sea r el mayor número natural tal que $1 + 2 + \cdots + r \leq z$. Definimos $x = z - (1 + 2 + \cdots + r) \in \omega$. Observemos que $x \leq r$ ya que si $x > r$, $z - (1 + 2 + \cdots + r) > r$, lo cual implica que $z - (1 + 2 + \cdots + r) \geq r + 1$ y por lo tanto $z \geq 1 + 2 + \cdots + r + (r + 1)$ contradiciendo la definición de r. Sea $y = r - x \in \omega$. Veamos que $\boldsymbol{\varphi_2}(x, y) = z$. En efecto,

$$\begin{aligned} \boldsymbol{\varphi_2}(x,y) &= \tfrac{(x+y+1)(x+y)}{2} + x \\ &= \tfrac{(r+1)r}{2} + [z - (1 + 2 + \cdots + r)] \\ &= \tfrac{(r+1)r}{2} + [z - \tfrac{(r+1)r}{2}] \\ &= z \end{aligned}$$

Por lo tanto, $\boldsymbol{\varphi_2}$ es sobreyectiva.

Ahora veamos que $\boldsymbol{\varphi_2}$ es inyectiva. Para ello sean $x, y, z \in \omega$ tal que $\boldsymbol{\varphi_2}(x, y) = z$, es decir $z = \frac{(x+y)(x+y+1)}{2} + x = \frac{x^2+2xy+x+y+y^2+2x}{2} = \frac{(x+y)^2+3x+y}{2}$. Por lo tanto,

$$\begin{aligned} z = \frac{(x+y)^2 + 3x + y}{2} &\Rightarrow 2z = (x+y)^2 + 3x + y \\ &\Rightarrow 8z + 1 = (2x + 2y + 1)^2 + 8x \\ &\Rightarrow (2x + 2y + 1)^2 \leq 8z + 1 < (2x + 2y + 3)^2 \\ &\Rightarrow 2x + 2y + 1 \leq \sqrt{8z + 1} < 2x + 2y + 3 \\ &\Rightarrow 2x + 2y + 1 \leq \lfloor \sqrt{8z + 1} \rfloor < 2x + 2y + 3 \\ &\Rightarrow x + y + 1 \leq \lfloor \frac{\lfloor \sqrt{8z + 1} \rfloor + 1}{2} \rfloor < x + y + 2 \\ &\Rightarrow x + y + 1 = \lfloor \frac{\lfloor \sqrt{8z + 1} \rfloor + 1}{2} \rfloor \\ &\Rightarrow x + y = \lfloor \frac{\lfloor \sqrt{8z + 1} \rfloor + 1}{2} \rfloor \dot{-} 1 \end{aligned}$$

Tenemos el sistema de ecuaciones:

$$\begin{cases} 3x + y = 2z \dot{-} (\lfloor \frac{\lfloor \sqrt{8z+1} \rfloor + 1}{2} \rfloor \dot{-} 1)^2 \\ x + y = \lfloor \frac{\lfloor \sqrt{8z+1} \rfloor + 1}{2} \rfloor \dot{-} 1 \end{cases}$$

El determinante del sistema es $\begin{vmatrix} 3 & 1 \\ 1 & 1 \end{vmatrix} = 3 - 1 = 2 \neq 0$. Por consiguiente el sistema tiene una única solución y por lo tanto x, y son únicos. Con esto, concluimos que $\boldsymbol{\varphi_2}$ es inyectiva. Por lo tanto, $\boldsymbol{\varphi_2}$ es biyectiva.

Para ver que $\boldsymbol{\varphi_2^{-1}}$ es 2-calculable basta notar que los pasos que seguimos para demostrar que $\boldsymbol{\varphi_2}$ es sobreyectiva, se pueden replicar con un programa. Para más detalles vaya al apéndice.

(ii) Demostremos el resultado por inducción sobre $n \geq 2$. En efecto, para $n = 2$, es justo el resultado anterior. Supongamos que el resultado es cierto para n y veamos que se cumple para $n + 1$.

Sea $\boldsymbol{\varphi_{n+1}} : \omega^{n+1} \to \omega$ tal que $\boldsymbol{\varphi_{n+1}}(x_1, ..., x_{n+1}) = \boldsymbol{\varphi_2}(x_1, \boldsymbol{\varphi_n}(x_2, ..., x_{n+1}))$ donde $\boldsymbol{\varphi_n}$ es la biyección obtenida por la hipótesis inductiva. Es evidente que $\boldsymbol{\varphi_{n+1}}$ es biyectiva.

Notemos que $\boldsymbol{\varphi_{n+1}}$ es computable ya que es la composición de funciones computables.

Para demostrar que $\boldsymbol{\varphi_{n+1}^{-1}}$ es $n + 1$-calculable, notemos que dado un número natural podemos calcular de forma computable su imagen bajo la función $\boldsymbol{\varphi_2^{-1}}$. Al segundo elemento de la tupla obtenida le calculamos de forma computable su imagen bajo la función $\boldsymbol{\varphi_n^{-1}}$. La tupla obtenida es la imagen del número natural original bajo la función $\boldsymbol{\varphi_{n+1}^{-1}}$. Para ver más detalles vaya al apéndice.

(iii) Definamos $\boldsymbol{\psi} : \omega^{<\mathbb{N}} \to \omega$ tal que $\boldsymbol{\psi}(x_1, ..., x_n) = \boldsymbol{\varphi_2}(n - 1, \boldsymbol{\varphi_n}(x_1, ..., x_n))$ para $n \geq 1$, donde $\boldsymbol{\varphi_1}$ es la función identidad en ω.

Afirmamos que $\boldsymbol{\psi}$ es biyectiva. Veamos que $\boldsymbol{\psi}$ es inyectiva. Sean $x_1, ..., x_n, y_1, ..., y_m \in \omega$ tales que

$$\begin{aligned}\boldsymbol{\psi}(x_1, ..., x_n) = \boldsymbol{\psi}(y_1, ..., y_m) &\Rightarrow \boldsymbol{\varphi_2}(n - 1, \boldsymbol{\varphi_n}(x_1, ..., x_n)) = \boldsymbol{\varphi_2}(m - 1, \boldsymbol{\varphi_m}(y_1, ..., y_m)) \\ &\Rightarrow n - 1 = m - 1 \ \wedge \ \boldsymbol{\varphi_n}(x_1, ..., x_n) = \boldsymbol{\varphi_m}(y_1, ..., y_m) \\ &\Rightarrow \boldsymbol{\varphi_n}(x_1, ..., x_n) = \boldsymbol{\varphi_n}(y_1, ..., y_n) \\ &\Rightarrow (x_1, ..., x_n) = (y_1, ..., y_m)\end{aligned}$$

Ahora veamos que $\boldsymbol{\psi}$ es sobreyectiva. Sea $z \in \omega$. Por consiguiente existen $n, y \in \omega$ tal que $\boldsymbol{\varphi_2}(n, y) = z$. Para $n + 1 \geq 1$ y $y \in \omega$, existen $x_1, ..., x_{n+1} \in \omega$ tal que $\boldsymbol{\varphi_{n+1}}(x_1, ..., x_{n+1}) = y$. Observemos que $\boldsymbol{\psi}(x_1, ..., x_{n+1}) = \boldsymbol{\varphi_2}(n, \boldsymbol{\varphi_{n+1}}(x_1, ..., x_{n+1})) = \boldsymbol{\varphi_2}(n, y) = z$. Con esto concluimos que $\boldsymbol{\psi}$ es biyectiva.

Para demostrar que $\boldsymbol{\psi}$ es calculable, notemos que $\boldsymbol{\varphi_n}(x_1, ..., x_n) = \underbrace{\boldsymbol{\varphi_2}(x_1, \cdots \boldsymbol{\varphi_2}(x_{n-1}, x_n)))}_{n-1 veces}$. Por lo tanto podemos obtener el valor de $\boldsymbol{\varphi_n}$ en una tupla si conocemos cuántas veces aplicaremos $\boldsymbol{\varphi_2}$ a los elementos de una tupla. Lo cual implica que podemos obtener de forma computable el valor

de $\boldsymbol{\psi}$ en una tupla finita. Para más detalles, vea el apéndice.

Para ver que $\boldsymbol{\psi}^{-1}$ es calculable, aplicamos un procedimiento similar al de $\boldsymbol{\psi}$. Aplicamos $\boldsymbol{\varphi}_{\mathbf{2}}^{-1}$ de forma computable a un número natural y el primer elemento de la tupla obtenida es la cantidad de veces que aplicaremos $\boldsymbol{\varphi}_{\mathbf{2}}^{-1}$ al segundo elemento de las tuplas obtenidas que vayamos obteniendo. Al final devolvemos la tupla obtenida. Para más detalles, vea el apéndice.

□

Ahora asignaremos un número natural a una instrucción de forma biyectiva de modo que podamos asignar un número natural a un programa.

Definición 1.3.3. *(i)* Definimos $I : \mathbf{INST} \to \omega$ como la función tal que:

$$I(f) = \begin{cases} 8(n-1) & \text{si } f = Z_n \text{ para algún } n \in \mathbb{N} \\ 8(n-1)+1 & \text{si } f = S_n \text{ para algún } n \in \mathbb{N} \\ 8(n-1)+2 & \text{si } f = D_n \text{ para algún } n \in \mathbb{N} \\ 8(\boldsymbol{\varphi}_{\mathbf{2}}(m,n)-1)+3 & \text{si } f = T_{m,n} \text{ para algunos } m,n \in \mathbb{N} \\ 8(\boldsymbol{\varphi}_{\mathbf{3}}(m,n,s)-1)+4 & \text{si } f = J_{m,n,s} \text{ para algunos } m,n,s \in \mathbb{N} \\ 8(\boldsymbol{\varphi}_{\mathbf{2}}(m,n)-1)+5 & \text{si } f = L_{m,n} \text{ para algunos } m,n \in \mathbb{N} \\ 8(\boldsymbol{\varphi}_{\mathbf{2}}(m,n)-1)+6 & \text{si } f = G_{m,n} \text{ para algunos } m,n \in \mathbb{N} \\ 8(\boldsymbol{\varphi}_{\mathbf{2}}(m,n)-1)+7 & \text{si } f = Q_{m,n} \text{ para algunos } m,n \in \mathbb{N} \end{cases}$$

Si $I(f) = i$ para algún $i \in \omega$, decimos que i **es un código de** f. Diremos que I es una **función aritmetizadora de instrucciones**.

(ii) Dado $A \subseteq \omega$, definimos $I^A : \mathbf{INST}^A \to \omega$ como la función tal que:

$$I^A(f) = \begin{cases} 9(n-1) & \text{si } f = Z_n \text{ para algún } n \in \mathbb{N} \\ 9(n-1)+1 & \text{si } f = S_n \text{ para algún } n \in \mathbb{N} \\ 9(n-1)+2 & \text{si } f = D_n \text{ para algún } n \in \mathbb{N} \\ 9(\boldsymbol{\varphi}_{\mathbf{2}}(m,n)-1)+3 & \text{si } f = T_{m,n} \text{ para algunos } m,n \in \mathbb{N} \\ 9(\boldsymbol{\varphi}_{\mathbf{3}}(m,n,s)-1)+4 & \text{si } f = J_{m,n,s} \text{ para algunos } m,n,s \in \mathbb{N} \\ 9(\boldsymbol{\varphi}_{\mathbf{2}}(m,n)-1)+5 & \text{si } f = L_{m,n} \text{ para algunos } m,n \in \mathbb{N} \\ 9(\boldsymbol{\varphi}_{\mathbf{2}}(m,n)-1)+6 & \text{si } f = G_{m,n} \text{ para algunos } m,n \in \mathbb{N} \\ 9(\boldsymbol{\varphi}_{\mathbf{2}}(m,n)-1)+7 & \text{si } f = Q_{m,n} \text{ para algunos } m,n \in \mathbb{N} \\ 9(n-1)+8 & \text{si } f = O_n^A \text{ para algún } n \in \mathbb{N} \end{cases}$$

Si $I(f) = i$ para algún $i \in \omega$, decimos que i **es un código de** f **respecto al oráculo** A. Diremos que I^A es una **función aritmetizadora de instrucciones con oráculo** A

Observación. Dado $A \subseteq \omega$, las funciones I e I^A son biyectivas.

Demostración. Veamos que I es inyectiva. Sean $f, g \in \mathbf{INST}$ tales que $I(f) = I(g)$. Si $res(8, I(f)) \in \{0, 1, 2\}$ entonces $I(f) = 8(n-1) + res(8, I(f))$ y $I(g) = 8(n'-1) + res(8, I(g))$ para algunos $n, n' \in \mathbb{N}$. Como $res(8, I(f)) = res(8, I(g))$, tenemos que $n = n'$. Luego ambas instrucciones tienen los mismos índices. Y como $res(8, I(f)) = res(8, I(g))$, tenemos que $f = g$.

Si $res(8, I(f)) \in \{3, 5, 6, 7\}$ entonces $I(f) = 8(\boldsymbol{\varphi_2}(m, n) - 1) + res(8, I(f))$ y $I(g) = 8(\boldsymbol{\varphi_2}(m', n') - 1) + res(8, I(g))$. Además como $res(8, I(f)) = res(8, I(g))$ tenemos que $\boldsymbol{\varphi_2}(m, n) = \boldsymbol{\varphi_2}(m', n')$, lo cual implica que $m = m'$ y $n = n'$. Y como $res(8, I(f)) = res(8, I(g))$, tenemos que $f = g$.

Si $res(8, I(f)) = 4$ entonces $I(f) = 8(\boldsymbol{\varphi_3}(m, n, s) - 1) + 4$ y $I(g) = 8(\boldsymbol{\varphi_3}(m', n', s') - 1) + 4$, lo cual implica que $\boldsymbol{\varphi_3}(m, n, s) = \boldsymbol{\varphi_3}(m', n', s')$, es decir que $m = m', n = n'$ y $s = s'$. Lo cual implica que $f = g$.

Ahora veamos que I es sobreyectiva. Sea $z \in \omega$. Por lo tanto, existen $k, r \in \omega$ tal que $z = 8k + r$ con $0 \leq r < 8$. Si $r \in \{0, 1, 2\}$ entonces suponiendo que $r = 0$, tenemos que $f = Z_{k+1}$ es tal que $I(f) = 8(k + 1 - 1) = 8k = z$. De manera análoga se tiene si r es alguno de los demás números. Si $r \in \{3, 5, 6, 7\}$ entonces para $k + 1 \in \omega$ existen $m, n \in \omega$ tal que $\boldsymbol{\varphi_2}(m, n) = k + 1$. Por ejemplo si $r = 3$, entonces para $f = T_{m,n}$ tenemos que $I(f) = 8(\boldsymbol{\varphi_2}(m, n) - 1) + 3 = 8(k + 1 - 1) + 3 = z$. Lo mismo ocurre para $r = 7$. Si $r = 4$, tenemos un caso similar al anterior sólo que con $\boldsymbol{\varphi_3}$.

Concluimos que I es biyectiva. La demostración para I^A es similar. □

De esta forma, podemos definir una función de aritmetización de programas.

Definición 1.3.4. *(i)* Definimos $prog : \mathbf{PROG} \to \omega$ tal que para todo $\mathbf{P} = (\mathbf{I}_1, ..., \mathbf{I}_k) \in \mathbf{PROG}$, $prog(\mathbf{P}) = \boldsymbol{\psi}(I(\mathbf{I}_1), ..., I(\mathbf{I}_k))$. Diremos que $prog$ es una **función de aritmetización de programas RAM**.

(ii) Dado $A \subseteq \omega$, definimos $prog^A : \mathbf{PROG}^A \to \omega$ tal que para todo $\mathbf{P} = (\mathbf{I}_1, ..., \mathbf{I}_k) \in \mathbf{PROG}^A$, $prog^A(\mathbf{P}) = \boldsymbol{\psi}(I^A(\mathbf{I}_1), ..., I^A(\mathbf{I}_k))$. Diremos que $prog^A$ es una **función de aritmetización de programas RAM con oráculo** A.

Una observación inmediata es la siguiente:

Observación. Las funciones $prog$ y $prog^A$ son biyectivas.

Demostración. Se sigue de que tanto $\boldsymbol{\psi}$ como I, ó I^A, son funciones biyectivas. □

Por la observación anterior podemos concluir que existe la función inversa de $prog$, ó bien $prog^A$ respecto a un oráculo A.

Dicho esto, hagamos las siguientes convenciones en notación.

Notación. Sean $A \subseteq \omega$ y $e, n \in \omega$ con $n \geq 1$.

(i) Denotamos por $\mathbf{P}_e$ a $(prog)^{-1}(e)$. Y por $\mathbf{P}_e^A$ a $(prog^A)^{-1}(e)$.

(ii) A la función n-aria definida por $\mathbf{P}_e$, $\mathbf{P}_e^A$ respectivamente, la denotamos únicamente por Φ_e^n, $\Phi_e^{A,n}$ respectivamente.

En el caso de una variable, en lugar de Φ_e^1, $\Phi_e^{A,n}$ respectivamente, escribimos Φ_e, Φ_e^A respectivamente.

(iii) Entendemos al predicado $\Phi_e^n(x_1, ..., x_n) \downarrow$ que es verdadero si y sólo si $\mathbf{P}_e(x_1, ..., x_n) \downarrow$.

También $\Phi_e^n(x_1, ..., x_n) \downarrow y$ es verdadero si y sólo si $\mathbf{P}_e(x_1, ..., x_n) \downarrow y$ para algún $y \in \omega$. Omitimos a n cuando $n = 1$.

(iv) De la misma forma el predicado $\Phi_e^{A,n}(x_1, ..., x_n) \downarrow$ es verdadero si y sólo si $\mathbf{P}_e^A(x_1, ..., x_n) \downarrow$.
También $\Phi_e^{A,n}(x_1, ..., x_n) \downarrow y$ es verdadero si y sólo si $\mathbf{P}_e^A(x_1, ..., x_n) \downarrow y$ para algún $y \in \omega$. Omitimos a n cuando $n = 1$.

(v) El predicado $\Phi_e^n(x_1, ..., x_n) \uparrow$ es verdadero si y sólo si $\mathbf{P}_e(x_1, ..., x_n) \uparrow$. Y $\Phi_e^{A,n}(x_1, ..., x_n) \uparrow$ es verdadero si y sólo si $\mathbf{P}_e^{A,n}(x_1, ..., x_n) \uparrow$. Quitamos a n cuando $n = 1$.

(vi) Para $s \in \mathbb{N}$, decimos que $\Phi_{e,s}^n(x_1, ..., x_n) \downarrow$ es verdadero si y sólo si $\mathbf{P}_e^n(x_1, ..., x_n) \downarrow$ implica que el tiempo de ejecución de $\mathbf{P}_e^n$ en $(x_1, ..., x_n)$ es menor o igual a s.

También $\Phi_{e,s}^n(x_1, ..., x_n) \downarrow y$ es verdadero si y sólo si $\Phi_{e,s}^n(x_1, ..., x_n) \downarrow$ y $\mathbf{P}_e^n(x_1, ..., x_n) \downarrow y$ para algún $y \in \omega$.

(vii) De manera análoga, para $s \in \mathbb{N}$, decimos que $\Phi_{e,s}^{A,n}(x_1, ..., x_n) \downarrow$ es verdadero si y sólo si $\mathbf{P}_e^{A,n}(x_1, ..., x_n) \downarrow$ implica que el tiempo de ejecución de $\mathbf{P}_e^{A,n}$ en $(x_1, ..., x_n)$ es menor o igual a s.

También $\Phi_{e,s}^{A,n}(x_1, ..., x_n) \downarrow y$ es verdadero si y sólo si $\Phi_{e,s}^{A,n}(x_1, ..., x_n) \downarrow$ y $\mathbf{P}_e^{A,n}(x_1, ..., x_n) \downarrow y$ para algún $y \in \omega$. Omitimos a n cuando $n = 1$.

(viii) Diremos que $\Phi_{e,s}^n(x_1, ..., x_n) \uparrow$ es verdadero si y sólo si $\mathbf{P}_e^n(x_1, ..., x_n) \downarrow$ y el tiempo de ejecución es mayor a s ó $\mathbf{P}_e^n(x_1, ..., x_n) \uparrow$.

Y $\Phi_{e,s}^{A,n}(x_1, ..., x_n) \uparrow$ es verdadero si y sólo si $\mathbf{P}_e^{A,n}(x_1, ..., x_n) \downarrow$ y el tiempo de ejecución es mayor a s ó $\mathbf{P}_e^{A,n}(x_1, ..., x_n) \uparrow$. Omitimos a n cuando $n = 1$.

A continuación una definición

Definición 1.3.5. Dados $\sigma \in \omega^{<\mathbb{N}}$, $n, e, s \in \omega$ con $n, s \geq 1$.

(i) El predicado $\Phi^{\sigma,n}_{e,s}(x_1,...,x_n)\downarrow$ es verdadero si y sólo si existe $A \subseteq \omega$ tal que para todo $x < len(\sigma)$, $\sigma(x) = \chi_A(x)$, toda instrucción oráculo de $\mathbf{P}^{A,n}_e$ tiene índice menor que la longitud de σ y $\Phi^{A,n}_{e,s}(x_1,...,x_n)\downarrow$.

Decimos que $\Phi^{\sigma,n}_{e,s}(x_1,...,x_n)\downarrow y$ para algún $y \in \omega$ si además $\Phi^{A,n}_{e,s}(x_1,...,x_n)\downarrow y$.

(Ii) Diremos que $\Phi^{\sigma,n}_{e}(x_1,...,x_n)\downarrow$ es verdadero si y sólo si existe $t \in \mathbb{N}$ tal que $\Phi^{\sigma,n}_{e,t}(x_1,...,x_n)\downarrow$.

También $\Phi^{\sigma,n}_{e}(x_1,...,x_n)\downarrow y$ para algún $y \in \omega$ si y sólo si existe $t \in \mathbb{N}$ tal que $\Phi^{\sigma,n}_{e,t}(x_1,...,x_n)\downarrow y$.

Notación. Sean $e, n, s \in \omega$ con $n, s \geq 1$, $\sigma \in \omega^{<\mathbb{N}}$ y $A \subseteq \omega$.

(i) Definimos $W^n_e := \text{dom } \Phi^n_e$; $W^{A,n}_e := \text{dom } \Phi^{A,n}_e$. Omitimos a n cuando $n = 1$.

(ii) Denotamos $W^n_{e,s} := \{(x_1,...,x_n) : \Phi^n_{e,s}(x_1,...,x_n)\downarrow\}$. Y al conjunto $W^{A,n}_{e,s} := \{(x_1,...,x_n) : \Phi^{A,n}_{e,s}(x_1,...,x_n)\downarrow\}$. Omitimos a n cuando $n = 1$.

(iii) Definimos $W^{\sigma,n}_{e,s} := \{(x_1,...,x_n) : \Phi^{\sigma,n}_{e,s}(x_1,...,x_n)\downarrow\}$. Y al conjunto $W^{\sigma,n}_{e} := \{(x_1,...,x_n) : \Phi^{\sigma,n}_{e}(x_1,...,x_n)\downarrow\}$. Omitimos a n cuando $n = 1$.

Lema 1.3.1. *Para cualesquiera $n \in \omega$ con $n \geq 1$ y $A \subseteq \omega$.*

(i) Existe una función $k : \omega \to \omega$ tal que $k(e)$ es un número natural tal que $\Phi_{k(e)} \circ \boldsymbol{\varphi_n} = \Phi^n_e$.

(ii) Existe una función $k : \omega \to \omega$ tal que $k(e)$ es un número natural tal que $\Phi^A_{k(e)} \circ \boldsymbol{\varphi_n} = \Phi^{A,n}_e$.

Demostración. *(i)* Sea $e \in \omega$ y sea $\mathbf{Q}$ el programa que hace a $\boldsymbol{\varphi}^{-1}_{\boldsymbol{n}}$ n-calculable. Sea $e_0 \in \omega$, tal que $\mathbf{Q}_{e_0}$ es el programa que primero calcula de forma computable $\boldsymbol{\varphi}^{-1}_{\boldsymbol{n+1}}$ y al resultado obtenido le aplica $\mathbf{P}_e$.

Dado $x \in \omega$, si $\boldsymbol{\varphi}^{-1}_{\boldsymbol{n}}(x) \in \text{dom } \Phi^n_e$ entonces $\Phi_{e_0}(x) = \Phi^n_e(\boldsymbol{\varphi}^{-1}_{\boldsymbol{n}}(x))$, es decir que $\Phi_{e_0} = \Phi^n_e \circ \boldsymbol{\varphi}^{-1}_{\boldsymbol{n}}$, lo cual implica que $\Phi_{e_0} \circ \boldsymbol{\varphi_n} = \Phi^n_e$. Es evidente que si $\boldsymbol{\varphi}^{-1}_{\boldsymbol{n}}(x) \notin \text{dom } \Phi^n_e$ entonces $x \notin \text{dom } \Phi_{e_0}$. De este modo, $\Phi_{e_0} \circ \boldsymbol{\varphi_n} = \Phi^n_e$. Definimos $k(e) := e_0$.

(ii) La demostración es similar.

□

Del resultado anterior, nuestro estudio únicamente se restringe a funciones de una sola variable. Más precisamente tenemos lo siguiente:

Teorema 1.3.1. *(i) Una función parcial $f : \omega^n \rightharpoonup \omega$ es parcialmente computable si y sólo si existe $e \in \omega$ tal que $f = \Phi_e \circ \boldsymbol{\varphi_n}$.*

(ii) Una función parcial $f : \omega^n \rightharpoonup \omega$ es parcialmente A-computable si y sólo si existe $e \in \omega$ tal que $f = \Phi_e^A \circ \boldsymbol{\varphi_n}$.

Demostración. Se sigue de las definiciones y del resultado anterior. □

De esta forma, todas las propiedades que demostremos de aquí en adelante serán únicamente para funciones de una sola variable y sobre conjuntos de ω (sus funciones características son de una sola variable), extendiéndose de la forma descrita en el resultado anterior.

Capítulo 2

Resultados importantes de la Teoría de la Computabilidad

2.1. Existencia de los programas universales

A partir del hecho de que hay una cantidad numerable de programas y una cantidad numerable de programas con oráculo A, surge la siguiente observación.

Observación. Existe una cantidad numerable de funciones de ω en ω que son computables. También, dado un oráculo $A \subseteq \omega$, existe una cantidad numerable de funciones de ω en ω que son A-computables.

Demostración. Por el teorema 1,3,1, existe una función F que manda a una función computable f a su número $e \in \omega$. Más aún esta función es inyectiva. En efecto, si $F(f) = F(g)$ entonces $f = \Phi_{F(f)} = \Phi_{F(g)} = g$. Por lo tanto, el conjunto de funciones computables es a lo más numerable. No obstante, dicho conjunto es infinito ya que toda función constante es computable, es decir dado $n \in \omega$ la función $c : \omega \to \omega$ tal que $c(x) = n$ para todo $x \in \omega$, es computable. Por lo tanto, hay una cantidad numerable de funciones computables. La demostración para un oráculo $A \subseteq \omega$ es similar. □

Esto nos permite deducir lo siguiente.

Observación. Existe una cantidad numerable de conjuntos de ω que son computables. Por lo tanto, hay conjuntos de ω que no son computables más aún existen $2^{\aleph_0}$ de estos conjuntos que no son computables.

Demostración. Existen $2^{\aleph_0}$ conjuntos de números naturales, mismos que podemos identificar con sus respectivas funciones características, χ_A. Luego, como hay una cantidad numerable de funciones computables, debe haber una cantidad a lo más numerable de funciones características que son computables.

Además como hay una cantidad numerable de conjuntos finitos y cada conjunto finito es computable entonces debe haber al menos una cantidad infinita de funciones características computables. Por consiguiente, hay una cantidad numerable de conjuntos de números naturales que son computables.

Como ω es un cardinal estrictamente menor que $2^{\aleph_0}$, se sigue que existe al menos un conjunto que no es computable, más aún utilizando resultados de la teoría de conjuntos tenemos que existen $2^{\aleph_0}$ conjuntos que no son computables. □

En este capítulo nos encargaremos de mostrar explícitamente un conjunto que no es computable. Y en general, dado $A \subseteq \omega$, describiremos cómo es un conjunto que no es A-computable.

Para ello, primero mostraremos la existencia de un programa que es capaz de producir el resultado de cualquier programa en cualquier entrada siempre y cuando este programa se detenga en dicha entrada, en este sentido, definiremos un *programa universal*.

Definición 2.1.1. *(a)* Denotamos por $\Psi_U : \omega \times \omega \nrightarrow \omega$ a la función parcial tal que:

$$\Psi_U(e,x) = \begin{cases} \Phi_e(x) & \text{si} \quad \Phi_e(x)\downarrow \\ indefinida & \text{en otro caso} \end{cases}$$

(b) Dado $A \subseteq \omega$, denotamos por $\Psi_U^A : \omega \times \omega \nrightarrow \omega$ a la función parcial tal que:

$$\Psi_U^A(e,x) = \begin{cases} \Phi_e^A(x) & \text{si} \quad \Phi_e^A(x)\downarrow \\ indefinida & \text{en otro caso} \end{cases}$$

Notemos que Ψ_U, o bien Ψ_U^A, es en efecto una función parcial ya que existen programas que no se detienen en ciertas entradas, más aún estos pueden no detenerse en ninguna. También notemos que ésta función parcial es capaz de devolver el resultado de una programa en una entrada siempre y cuando esté último se encuentre definido. En este sentido, las funciones Ψ_U y Ψ_U^A son universales.

En lo sucesivo, nos encargaremos de demostrar que la función Ψ_U es parcialmente computable (la demostración para la función Ψ_U^A es similar). Informalmente bastaría con definir un programa $\mathbf{P}$ que en una entrada (e,x) decodifique el programa número e y posteriormente lea, reconozca y ejecute las instrucciones obtenidas en la entrada x de la misma forma que lo haría el programa $\mathbf{P}_e$ en dicha entrada. Es evidente, que este programa se detendría si y sólo si el programa número e se detiene y en caso de detenerse su resultado coincidiría con el del programa e en la entrada x. Sin embargo, por la importancia del resultado daremos una demostración formal del argumento anterior.

Para ello mostremos antes los siguientes resultados:

Lema 2.1.1. *Las siguientes funciones son (parcialmente) computables según sea el caso:*

(i) *La función* $inst : \omega \to \omega$ *tal que*

$$inst(e) = \textit{el número de instrucciones del programa } \mathbf{P}_e$$

(ii) *La función* $cod : \omega \times \omega \to \omega$ *tal que*

$$cod(e,k) = \begin{cases} \textit{el código de la instrucción número } k \textit{ en } \mathbf{P}_e & \textit{si} \quad 1 \leq k \leq inst(e) \\ 0 & \textit{en otro caso} \end{cases}$$

(iii) *La función parcial* $cmod : \omega \times \omega \nrightarrow \omega$ *tal que*

$$cmod(c,z) \;=\; \textit{la configuración resultante tras ejecutar la instrucción cuyo código es } z \textit{ en la configuración } c$$

(iv) *La función parcial* $imod : \omega^3 \nrightarrow \omega$ *tal que*

$$imod(c,j,z) \;=\; \begin{cases} \textit{el número de la instrucción que se obtiene tras ejecutar la instrucción de código } z \textit{ y número } j \textit{ en la configuración } c & \textit{si} \quad j > 0 \\ 0 & \textit{en otro caso} \end{cases}$$

(v) *La función parcial* $config : \omega^3 \nrightarrow \omega$ *tal que*

$$config(e,c,j) \;=\; \begin{cases} \textit{la configuración resultante tras ejecutar la instrucción número } j \textit{ del programa } \mathbf{P}_e \textit{ a la configuración } c & \textit{si } 1 \leq j \leq inst(e) \\ c & \textit{en otro caso} \end{cases}$$

(vi) *La función parcial* $siginst : \omega^3 \nrightarrow \omega$ *tal que*

$$siginst(e,c,j) \;=\; \begin{cases} \textit{el número de la instrucción que se obtiene tras ejecutar la instrucción número } j \textit{ del programa } \mathbf{P}_e \textit{ a la configuración } c & \textit{si } 1 \leq j \leq inst(e) \textit{ y la instrucción obtenida forma parte del programa } \mathbf{P}_e \\ 0 & \textit{en otro caso} \end{cases}$$

Demostración. Demostraremos el inciso *(a)* siendo que la demostración del inciso *(b)* es análoga.

(i) Sea $\mathbf{Q}_1$ el programa que hace a $\boldsymbol{\varphi}_{\mathbf{2}}^{-1}$ 2-calculable. Entonces $inst(e) = \pi_1(\boldsymbol{\varphi}_{\mathbf{2}}^{-1}(e)) + 1$, donde $\pi_1 : \omega^{<\mathbb{N}} \to \omega$ tal que $\pi_1(x_1, ..., x_n) = x_1$. Por lo tanto, $inst(e) = \Phi_{\mathbf{Q}_1}(e) + 1$, de lo cual podemos concluir que $inst$ es una función computable.

(ii) Sea $\mathbf{Q}_1$ el programa que hace a $\boldsymbol{\psi}^{-1}$ calculable con la modificación de que antes de iniciar el ciclo de dicho programa agregamos la instrucción que manda a lo que se encuentra en r_{10} al final de la tupla que vayamos a obtener, es decir agregamos las instrucciones que conforman a $r_{12+r_9+1} \leftarrow r_{10}$ y las modificaciones necesarias a las instrucciones de salto del programa en general. Sea $\mathbf{Q}$ el siguiente programa.

Programa en pseudocódigo	**Explicación del programa**
$r_{10} \leftarrow r_2$	Guardamos la segunda coordenada de la entrada, cuyo valor es k y lo guardamos en el registro r_{10} para posteriormente utilizarlo para devolver la instrucción del programa e.
$r_2 \leftarrow 0$	Hacemos cero el registro r_2 para utilizarlo después.
$\mathbf{Q}_1$	Se ejecuta el programa de $\boldsymbol{\psi}^{-1}$ con las modificaciones mencionadas con anterioridad. El resultado que obtenemos es $\boldsymbol{\psi}^{-1}(e)$ si este existe y al final de la tupla finita el valor de k.
si $0 < r_{12+r_9+1} \leq r_9$ **entonces**	Si $1 \leq k \leq len(\boldsymbol{\psi}^{-1}(e))$ entonces...
$r_1 \leftarrow r_{12+r_{12+r_9+1}}$	Mandamos al registro r_1 el valor que se encuentra en la posición k-ésima de la tupla que obtuvimos. Esto con motivo de que sea el resultado final del programa. Dicho valor corresponde al código de la k-ésima instrucción de $\mathbf{P}_e$.

en otro caso

$r_1 \leftarrow 0$	En caso de que sea falso que $1 \leq k \leq len(\boldsymbol{\psi}^{-1}(e))$ entonces únicamente hacemos 0 el valor del registro r_1 para que sea el resultado final que va a devolver el programa.

fin si

Como hemos visto en otros programas escritos en pseudocódigo, éste programa también se puede escribir con instrucciones RAM.

Ahora bien si $e, k \in \omega$, con $1 \leq k \leq inst(e)$, entonces $cod(e,k) = \pi_k(\boldsymbol{\psi}^{-1}(e)) = \Phi^2_{\mathbf{Q}}(e,k)$, donde $\pi_k : \omega^{<\mathbb{N}} \to \omega$ es tal que $\pi_k(x_1, ..., x_n) = x_k$ si $1 \leq k \leq n$ e indefinida en otro caso. Si no ocurre que $1 \leq k \leq inst(e)$ entonces $cod(e,k) = 0 = \Phi^2_{\mathbf{Q}}$. Por lo tanto, $cod = \Phi^2_{\mathbf{Q}}$. Lo cual nos permite concluir que cod es una función computable.

(iii) Demostraremos este inciso definiendo para cada tipo de instrucciones posibles una función parcialmente computable que nos da la configuración resultante de aplicar esa instrucción a la tupla finita de números naturales que le corresponde a una configuración. Dada una configuración c, denotamos por $j = \min_{x \in \omega}\{exp(c,x) = 0\}$. Observemos que j se puede interpretar como el menor índice de los registros que no se encuentran en el contenido útil de una máquina de acceso aleatorio. También recordemos que el contenido útil de una máquina de acceso aleatorio lo consideramos como una tupla finita de la forma $(r_1, ..., r_{j-1})$, luego su configuración es $\prod_{i=1}^{j-1} p(i)^{r_i+1}$. A partir de esto notemos que dada una configuración podemos obtener de forma computable los valores $r_1, ..., r_{j-1}$ utilizando un algoritmo de factorización en números primos. Dicho esto, continuemos con la demostración.

Instrucciones cero

$$Z(c,z) = \begin{cases} qt(p(k)^{r_k}, c) & \text{si } res(8,z) = 0 \text{ y } 1 \leq k < j \\ & \text{donde } k = qt(8,z) + 1 \\ c \cdot \prod_{i=j}^{k} p(i) & \text{si } res(8,z) = 0 \text{ y } j \leq k \\ & \text{donde } k = qt(8,z) + 1 \\ c & \text{si ocurre otro caso} \end{cases}$$

Instrucciones sucesor

$$S(c,z) = \begin{cases} cp(k) & \text{si } res(8,z) = 1 \text{ y } 1 \leq k < j \\ & \text{donde } k = qt(8, z \dot{-} 1) + 1 \\ c \cdot (\prod_{i=j}^{k-1} p(i))p(k)^2 & \text{si } res(8,z) = 1 \text{ y } j \leq k \\ & \text{donde } k = qt(8, z \dot{-} 1) + 1 \\ c & \text{si ocurre otro caso} \end{cases}$$

Instrucciones de salto

$$J(c,z) = c$$

Instrucciones de transferencia

$$T(c,z) = \begin{cases} qt(p(s)^{r_s+1}, c)p(s)^{r_k+1} & \text{si } res(8,z) = 3 \text{ y } k, s < j \text{ donde} \\ & l = qt(8, z \dot{-} 3) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ & \text{y } s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ c \cdot (\prod_{i=j}^{s-1} p(i))p(s)^{r_k+1} & \text{si } res(8,z) = 3 \text{ y } k < j < s \text{ donde} \\ & l = qt(8, z \dot{-} 3) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ & \text{y } s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ c \cdot p(s)^{r_k+1} & \text{si } res(8,z) = 3 \text{ y } k < j = s \text{ donde} \\ & l = qt(8, z \dot{-} 3) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ & \text{y } s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ qt(p(s)^{r_s+1}, c)p(s) & \text{si } res(8,z) = 3 \text{ y } s < j \leq k \text{ donde} \\ & l = qt(8, z \dot{-} 3) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ & \text{y } s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ c & \text{si ocurre otro caso} \end{cases}$$

Instrucciones decremento

$$D(c,z) = \begin{cases} qt(p(k),c) & \text{si } res(8,z) = 2 \text{ y } 1 \leq k < j \text{ donde} \\ & k = qt(8, z \dot{-} 2) + 1 \text{ y } exp(c,k) > 1 \\ c \cdot \prod_{i=j}^{k} p(i) & \text{si } res(8,z) = 2 \text{ y } j \leq k \\ & \text{donde } k = qt(8, z \dot{-} 2) + 1 \\ c & \text{si ocurre otro caso} \end{cases}$$

Instrucciones suma

$$Q(c,z) = \begin{cases} c \cdot p(s)^{r_k} & \text{si } res(8,z) = 7 \text{ y } 1 \leq k, s < j \text{ donde} \\ & l = qt(8, z \dot{-} 7) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ & \text{y } s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ c \cdot (\prod_{i=j}^{s-1} p(i)) p(s)^{r_k+1} & \text{si } res(8,z) = 7 \text{ y } 1 \leq k < j \leq s \text{ donde} \\ & l = qt(8, z \dot{-} 7) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ & \text{y } s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \\ c & \text{si ocurre otro caso} \end{cases}$$

Instrucciones de almacenamiento

$$G(c,z) = \begin{cases} qt(p(t)^{r_t+1}, c) p(t)^{r_s+1} & \text{si } res(8,z) = 6 \text{ y } 1 \leq k, s, t < j \text{ donde} \\ & l = qt(8, z \dot{-} 6) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ , } \\ & s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ y } t = exp(c,k) \dot{-} 1 \\ c \cdot (\prod_{i=j}^{t-1} p(i)) p(t)^{r_s+1} & \text{si } res(8,z) = 6 \text{ y } 1 \leq k, s < j \leq t \\ & \text{y } l = qt(8, z \dot{-} 6) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ , } \\ & s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ y } t = exp(c,k) \dot{-} 1 \\ qt(p(t)^{r_t+1}, c) p(t) & \text{si } res(8,z) = 6 \text{ y } 1 \leq k, t < j \leq s \\ & \text{y } l = qt(8, z \dot{-} 6) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ , } \\ & s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ y } t = exp(c,k) \dot{-} 1 \\ c & \text{si ocurre otro caso} \end{cases}$$

Instrucciones de carga

$$L(c,z) = \begin{cases} qt(p(s)^{r_s+1}, c)p(s)^{r_t+1} & \text{si } res(8,z) = 5 \text{ y } 1 \leq k,s,t < j \text{ donde} \\ & l = qt(8, z \dot{-} 5) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ ,} \\ & s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ y } t = exp(c,k) \dot{-} 1 \\ qt(p(s)^{r_s+1}, c)p(s) & \text{si } res(8,z) = 5 \text{ y } 1 \leq k,s < j \leq t \\ & \text{y } l = qt(8, z \dot{-} 5) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ , } \\ & s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ y } t = exp(c,k) \dot{-} 1 \\ c \cdot \prod_{i=j}^{s-1} p(i) \cdot p(s)^{r_t+1} & \text{si } res(8,z) = 5 \text{ y } 1 \leq k,t < j \leq s \\ & \text{y } l = qt(8, z \dot{-} 5) + 1 \text{ , } k = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ , } \\ & s = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(l)) \text{ y } t = exp(c,k) \dot{-} 1 \\ c & \text{si ocurre otro caso} \end{cases}$$

Para el caso en el que tengamos un oráculo A, las anteriores definiciones sólo cambian 8 por 9 y consideramos la siguiente función.

Instrucciones oráculo

$$O^A(c,z) = \begin{cases} qt(p(k)^{r_k+1}, c)[\chi_A(s)p(k)^2 + (1 - \chi_A(s)p(k))] & \text{si } res(9,z) = 8 \text{ y} \\ & 1 \leq k < j \text{ donde} \\ & k = qt(9, z \dot{-} 8) + 1 \\ & \text{y } s = exp(c,k) \dot{-} 1 \\ c \cdot (\prod_{i=j}^{s-1} p(i))[\chi_A(s)p(k)^2 + (1 - \chi_A(s)p(k))] & \text{si } res(9,z) = 8 \text{ y} \\ & j < k \text{ donde} \\ & k = qt(9, z \dot{-} 8) + 1 \\ & \text{y } s = exp(c,k) \dot{-} 1 \\ c \cdot [\chi_A(s)p(k)^2 + (1 - \chi_A(s)p(k))] & \text{si } res(9,z) = 8 \text{ y} \\ & j = k \text{ donde} \\ & k = qt(9, z \dot{-} 8) + 1 \text{ y} \\ & s = exp(c,k) \dot{-} 1 \\ c & \text{si ocurre otro caso} \end{cases}$$

Observemos que la función

$$cmod(c,z) = \begin{cases} Z(c,z) & \text{si } res(8,z) = 0 \\ S(c,z) & \text{si } res(8,z) = 1 \\ D(c,z) & \text{si } res(8,z) = 2 \\ T(c,z) & \text{si } res(8,z) = 3 \\ J(c,z) & \text{si } res(8,z) = 4 \\ L(c,z) & \text{si } res(8,z) = 5 \\ G(c,z) & \text{si } res(8,z) = 6 \\ Q(c,z) & \text{si } res(8,z) = 7 \end{cases}$$

Notemos que para verificar que cada función definida anteriormente es parcialmente computable, basta ver que cada una de las condiciones que la definen se pueden verificar y los valores de las funciones se pueden obtener de forma computable.

(iv) Sea $t = \min_{x\in\omega}\{exp(c,x) = 0\}$. Notemos que la función:

$$imod(c,j,z) = \begin{cases} l & \begin{array}{c}\text{si } j > 0 \text{ y } res(8,z) = 4 \\ \text{y } (m,n < t \text{ y } exp(c,m) = exp(c,n)) \\ \text{ó } (m < t \leq n \text{ y } exp(c,m) = 1) \text{ ó} \\ (n < t \leq m \text{ y } exp(c,n) = 1) \text{ ó} \\ (t < n,m) \\ \text{donde } k = qt(8, z\dot{-}4) + 1,\ m = \pi^3_1(\boldsymbol{\varphi_3}^{-1}(k)), \\ n = \pi^3_2(\boldsymbol{\varphi_3}^{-1}(k)) \text{ y } l = \pi^3_3(\boldsymbol{\varphi_3}^{-1}(k)) \end{array} \\ j+1 & \begin{array}{c}\text{si } j > 0 \text{ y ocurre cualquier otro caso} \\ \text{distinto a la condición anterior}\end{array} \\ 0 & \text{si } j \leq 0 \end{cases}$$

Similar a las funciones del inciso *(iii)*, es posible verificar que esta función es parcialmente computable verificando cada una de las condiciones y obteniendo los valores de la función de manera computable.

(v) Observemos que:

$$\begin{aligned} config(e,c,j) = \quad & sg(j)sg(inst(e)+1\dot{-}j)cmod(c,cod(e,j))+ \\ & +(1\dot{-}sg(j)sg(inst(e)+1\dot{-}j))c \end{aligned}$$

Como la función se compone de funciones que son parcialmente computables mediante operaciones de suma, resta, multiplicación y composición entonces es parcialmente computable.

(vi) Notemos que:

$$\begin{aligned} siginst(e,c,j) = \quad & sg(inst(e)+1\dot{-}j)\cdot sg(inst(e)+1\dot{-}imod(c,j,cod(e,j)))\cdot \\ & \cdot imod(c,j,cod(e,j)) \end{aligned}$$

Por una razón similar a la del inciso anterior tenemos que esta función es parcialmente computable.

El caso relativo a un oráculo $A \subseteq \omega$ es similar, únicamente añadimos a la función $O^A(c,z)$.

□

Lema 2.1.2. *Las siguientes funciones son computables:*

(i) La función $con : \omega^3 \to \omega$ *tal que*

$$con(e,x,t) = \textit{configuración de } \mathbf{P}_e(x) \textit{ después de } t \textit{ pasos}$$

(ii) La función $inst : \omega^3 \to \omega$ *tal que*

$$inst(e,x,t) = \begin{cases} \textit{el número de la instrucción obtenida tras ejecutarse exactamente } t \textit{ pasos en } \mathbf{P}_e(x) & \textit{si } \mathbf{P}_e(x) \textit{ no se ha detenido tras } t \textit{ pasos} \\ 0 & \textit{en otro caso} \end{cases}$$

(iii) La función $est : \omega^3 \to \omega$ *tal que*

$$est(e,x,t) = \boldsymbol{\varphi_2}(con(e,x,t), inst(e,x,t))$$

Esta función representa el estado de la máquina.

Demostración. Demostraremos que se cumple el inciso *(iii)*. Observemos que:

$$\begin{aligned} est(e,x,0) &= \boldsymbol{\varphi_2}(2^{x+1},1) \\ est(e,x,t+1) &= \boldsymbol{\varphi_2}(config(e,con(e,x,t),inst(e,x,t)), \\ & \quad siginst(e,con(e,x,t),inst(e,x,t))) \end{aligned}$$

donde

$$con(e,x,t) = \pi_1^2(\boldsymbol{\varphi_2}^{-1}(est(e,x,t))) \text{ y}$$
$$inst(e,x,t) = \pi_2^2(\boldsymbol{\varphi_2}^{-1}(est(e,x,t)))$$

Por lo que la función *est* se puede derfinir por recursión primitiva en base a su valor inicial y a los valores que vaya obteniendo. Por lo tanto la función *est* es computable, lo cual prueba el inciso *(iii)*.

Como podemos escribir a *con* y a *inst* en términos de *est* mediante funciones computables y calculables, entonces $config$ y *est* son computables, lo cual prueba los incisos *(i)* y *(ii)*.

□

Teorema 2.1.1. *(i) La función Ψ_U es parcialmente computable.*

(ii) Para cualquier $A \subseteq \omega$, Ψ_U^A es parcialmente A-computable.

Demostración. *(i)* Notemos que

$$\Psi_U(e,x) = exp(con(e,x,\min_{t\in\omega}\{inst(e,x,t) = 0\}),1) \dot{-} 1$$

Por consiguiente, concluimos que Ψ_U es una función parcialmente computable.

(ii) La demostración es similar definiendo los equivalentes de cada una de las funciones *cmod*, *con* e *inst*, las cuales distinguiremos con el superíndice A.

□

En la definición de la función parcial Ψ_U en la prueba anterior, notemos que la razón por la que no es total es que busca el menor instante de tiempo en el que el programa de número e se detiene en una determinada entrada. Como un programa puede o no detenerse en una entrada arbitraria, dicho instante t puede nunca encontrarse, lo cual hace imposible a la función *con* calcular la configuración final del cómputo y por consiguiente no se pueda obtener el resultado final. En resumen, que Ψ_U se encuentre definida en una entrada (e,x) depende única y exclusivamente de si el programa $\mathbf{P}_e$ se detiene en la entrada x.

Observemos que la definición de la función Ψ_U es explícita y en base a funciones que son computables. Por consiguiente, a partir de los códigos de los programas de estas funciones podemos obtener el código de la función Ψ_U, el cual por cuestiones de practicidad evitamos calcularlo explícitamente pero del cual haremos referencia cuando sea necesario.

Dicho esto, mostremos en general, dado un conjunto A, un ejemplo explícito de un conjunto que no es A-computable.

2.2. Ejemplos de conjuntos que no son computables

A partir de ahora, seremos más flexibles cuando mostremos que una función es (parcialmente A-) computable y daremos el algoritmo de manera informal, teniendo en cuenta que dicho algoritmo se puede escribir de manera formal con instrucciones RAM haciendo uso de la tesis de Church-Turing.

También a partir de ahora haremos uso de la observación que nos dice que una función es (parcialmente) computable si y sólo si es (parcialmente $\emptyset$-) computable, por lo que nuestros resultados serán respecto a cualquier oráculo $A \subseteq \omega$, en particular para $\emptyset$.

Teorema 2.2.1. *Sea $A \subseteq \omega$.*

(i) El conjunto $A' := \{x \in \omega : \Phi_x^A(x) \downarrow\}$ no es A-computable.

(ii) La función característica de A', $\chi_{A'}$, no es A-computable.

(iii) Sea $K^A := \{(e, x) \in \omega \times \omega : \Phi_e^A(x) \downarrow\}$. El conjunto $\boldsymbol{\varphi_2}[K^A]$ no es A-computable.

Demostración. *(i)* Supongamos que A' es A-computable. Afirmamos que la función $f^A : \omega \to \omega$ tal que:

$$f^A(x) = \begin{cases} \Phi_x^A(x) + 1 & \text{si} \quad \Phi_x^A(x) \downarrow \\ 0 & \text{si} \quad \text{ocurre otro caso} \end{cases}$$

es A-computable.
En efecto, dado $x \in \omega$, preguntamos si $x \in A'$. La respuesta la obtenemos de manera A-computable. En caso de que no, se asigna inmediatamente un 0 como salida del programa. Y en caso afirmativo, calculamos $\Psi_U^A(x, x) + 1$, que es lo mismo que $\Phi_x^A(x) + 1$. Entonces, como f^A es A-computable existe $e \in \omega$ tal que $f^A = \Phi_e^A$ y dom $\Phi_e^A = \omega$. Por consiguiente $\Phi_e^A(e) \downarrow$, lo cual implica que $\Phi_e^A(e) + 1 = f^A(e) = \Phi_e^A(e)$, que es una contradicción. Concluimos, entonces, que A' no es A-computable.

(ii) Es una consecuencia inmediata del inciso anterior.

(iii) Supongamos por el contrario que $\boldsymbol{\varphi_2}[K^A]$ es A-computable. Afirmamos que A' es A-computable. En efecto, tomemos $x \in \omega$ y verifiquemos si $(x, x) \in K^A$, o equivalentemente $\boldsymbol{\varphi_2}(x, x) \in \boldsymbol{\varphi_2}[K^A]$, con A como oráculo. En caso de que no entonces $x \notin A'$ y en caso contrario $x \in A'$. Por lo cual, concluimos que A' es A-computable, lo cual es falso. Por lo tanto, K^A no es A-computable.

□

Para ahorrar un poco la notación, en ocasiones consideraremos a K^A como un subcojunto de ω y en otras como un conjunto de pares ordenados, según convenga.

Enseguida mostraremos un resultado que nos ayudará a mostrar más ejemplos de conjuntos que no son A-computables.

Teorema 2.2.2. *(Teorema s-m-n) Para cada $n, m \geq 1$ existe una función computable e inyectiva $s_n^m : \omega^{m+1} \to \omega$ tal que para cada $e, x_1, .., x_m, y_1, ..., y_n \in \omega$ se cumple que:*

$$\Phi_e^{A,m+n}(x_1, ..., x_m, y_1, ..., y_n) = \Phi^{A,n}_{s_n^m(e,x_1,...,x_m)}(y_1, ..., y_n)$$

Demostración. Sean $m, n \in \omega$ con $n \geq 1$.

Sea $s_n^m : \omega^{m+1} \to \omega$ tal que para cada $e, x_1, ..., x_m \in \omega$ les asocia el código correspondiente al siguiente programa que recibe n entradas:

$$\begin{array}{l} r_{m+1} \leftarrow r_1 \\ r_{m+2} \leftarrow r_2 \\ \vdots \\ r_{m+n} \leftarrow r_n \\ r_1 \leftarrow x_1 \\ r_2 \leftarrow x_2 \\ \vdots \\ r_m \leftarrow x_m \\ \mathbf{P}_e^{A,m+n} \end{array}$$

El programa anterior toma una entrada de tamaño n, la recorre m lugares para colocar en ellos los números $x_1, ..., x_m$, para que finalmente ejecute el programa $\mathbf{P}_e^A$ en $m + n$ entradas.

Ahora notemos que dados $e, x_1, ..., x_n \in \omega$ podemos obtener el código del programa anterior de forma computable. En efecto, formamos una tupla finita con los códigos de las instrucciones del programa anterior. Las primeras n instrucciones son fijas y no dependen de ninguno de los números $e, x_1, ..., x_n$. Por lo tanto podemos hacer que el programa escriba dichos códigos. Posteriormente para cada x_i iniciamos un ciclo que tenga a i como contador que primero escriba el código de la instrucción Z_i y luego escriba x_i veces los códigos de S_i. La tupla de códigos obtenidos la concatenamos con la tupla anterior. Finalmente calculamos $\boldsymbol{\psi}^{-1}(e)$ y lo obtenido lo concatenamos con la tupla obtenida hasta el momento. Calculamos el valor bajo $\boldsymbol{\psi}$ de esta tupla y el resultado es el valor de $s_n^m(e, x_1, ..., x_n)$.

Para ver que s_n^m es inyectiva observemos que la construcción anterior nos daría dos tuplas finales distintas si partimos de dos tuplas distintas de la forma $(e, x_1, ..., x_n)$. Luego, como $\boldsymbol{\psi}$ es inyectiva tendríamos que los valores que obtengamos son distintos, lo cual implica que los valores bajo s_n^m también deben ser distintos. □

El resultado anterior nos dice que si tenemos un programa e y una entrada, podemos obtener de forma computable el programa que guarda parte de la entrada en su memoria y realiza lo mismo que el programa original.

A continuación mostraremos una consecuencia del teorema anterior, que nos será de mucha utilidad para probar que un conjunto no es A-computable.

Teorema 2.2.3. *(Teorema de Rice) Sean $A \subseteq \omega$ y $\mathcal{B} \subseteq \mathbf{PROG}^A$ con $\mathcal{B} \neq \emptyset$ y $\mathcal{B} \neq \mathbf{PROG}^A$. El conjunto $C^{\mathcal{B}} := \{e \in \omega : \mathbf{P}_e^A \in \mathcal{B}\}$ no es A-computable.*

Demostración. Sea $\mathcal{B} \subseteq \mathbf{PROG}^A$ como lo describe el resultado. Por contradicción supongamos que $C^{\mathcal{B}}$ es A-computable.
Sea $h : \omega \rightharpoonup \omega$ la función parcial cuyo dominio es el conjunto vacío, es decir que no está definida en ningún número natural. Sea e' el código de un programa que no se detiene en ninguna entrada. Es claro que $\Phi_{e'} = h$. Supongamos que $\mathbf{P}_{e'} \notin \mathcal{B}$. Sea $g : \omega \rightharpoonup \omega$ tal que $g = \Phi_{e_0}^A$ con $\mathbf{P}_{e_0}^A \in \mathcal{B}$ donde $e_0 = \min\limits_{k \in \omega}\{\mathbf{P}_k^A \in \mathcal{B}\}$. Dado que $C^{\mathcal{B}}$ es A-computable y es distinto del $\emptyset$, podemos obtener a e_0 de forma A-computable.

Definamos $f : \omega \times \omega \rightharpoonup \omega$ tal que

$$f(x,y) = \begin{cases} g(y) & si \quad \Phi_x^A(x)\downarrow \\ indefinida & si \quad \text{ocurre otro caso} \end{cases}$$

Notemos que el programa que en una entrada (x, y), primero calcula $\Phi_x^A(x)$ con las instrucciones de la función Ψ_U^A y en caso de obtener un resultado calcula $g(y)$ con las instrucciones del programa cuya función es Φ_{e_0}, es un programa que devuelve el valor de $f(x, y)$. El código de este programa se puede obtener de forma computable, ya que el código de Ψ_U^A es fijo y a la tupla de éste basta concatenar la tupla que bajo $\boldsymbol{\psi}$ es e_0. Sea r la función en una variable que es definida por este programa. Luego, $f = \Phi_{r(e_0)}$. Por el resultado anterior, existe $s : \omega \times \omega \to \omega$ función computable tal que $\Phi_{r(e_0)}(x, y) = \Phi_{s(r(e_0),x)}(y)$, es decir tal que $f(x, y) = \Phi_{s(r(e_0),x)}(y)$.

Por lo tanto, si $x \in A'$ entonces $\Phi_x(x) \downarrow$ lo cual implica que $\Phi_{s(r(e_0),x)} = g$ y $\mathbf{P}_{s(r(e_0),x)} \in \mathcal{B}$. Y si $x \notin A'$ entonces $\Phi_x(x) \uparrow$ lo cual implica que $\Phi_{s(r(e_0),x)} = h$ y $\mathbf{P}_{s(r(e_0),x)} \notin \mathcal{B}$. De modo que, $x \in A' \leftrightarrow \mathbf{P}_{s(r(e_0),x)} \in \mathcal{B}$. Es decir, $x \in A' \leftrightarrow s(r(e_0), x) \in C^{\mathcal{B}}$. Por consiguiente, dado $x \in \omega$, obtenemos $s(r(e_0), x)$ de forma A-computable y preguntamos si pertenece a $C^{\mathcal{B}}$, en caso de que sí, entonces $x \in A'$ y en caso contrario, $x \notin A'$. Por lo tanto, A' es A-computable, lo cual es una contradicción. Si $\mathbf{P}_{e'} \in \mathcal{B}$ la demostración es similar. □

Algunos ejemplos del resultado anterior son los siguientes:

Ejemplo 2.2.1. Sea $A \subseteq \omega$. Los siguientes conjuntos no son A-computables.

(i) $Symb^A := \{(e, x, s) \in \omega^3 : \exists t, r \in \omega\ (exp(con^A(e, x, t), r) \dot{-} 1 = s)\}$.

(ii) $DiagZ^A := \{x \in \omega : \Phi_x^A(x) \downarrow 0\}$.

(iii) El conjunto $\boldsymbol{\varphi_2}[Z^A]$ donde $Z^A := \{(e,x) \in \omega \times \omega : \Phi_e^A(x) \downarrow 0\}$.

(iv) $T^A := \{e \in \omega : \forall x \in \omega\ (\Phi_e^A(x) \downarrow)\}$.

(v) $Empty^A := \{e \in \omega : \forall x \in \omega\ (\Phi_e^A(x) \uparrow)\}$.

(vi) $E^A := \{x \in \omega : \exists y \in \omega\ (\Phi_x^A(y) = x)\}$.

(vii) $Range^A := \{e \in \omega : \forall n \in \omega\ (\exists x \in \omega(\Phi_e^A(x) \downarrow > n))\}$.

(viii) Sea $f : \omega \to \omega$ una función computable. El conjunto $Eq_f^A := \{e \in \omega : \Phi_e^A = f\}$ no es A-computable.

(ix) El conjunto $\boldsymbol{\varphi_2}[Sim^A]$ donde $Sim^A := \{(e_1,e_2) \in \omega \times \omega : \forall x \in \omega\ (\Phi_{e_1}^A(x) \downarrow \leftrightarrow \Phi_{e_2}^A(x) \downarrow)\}$.

Demostración. *(i)* Suponga que $Symb^A$ es A-computable. Sean $\mathcal{B} := \{\mathbf{P}_e^A : \exists t, r \in \omega\ (exp(con^A(e,e,t),r) \dot{-} 1 = 1)\} \subseteq \mathbf{PROG}^A$ y $One^A := \{e \in \omega : \mathbf{P}_e^A \in \mathcal{B}\}$. Ahora bien, notemos que $\mathcal{B} \neq \emptyset$ ya que el programa que sin importar la entrada escribe un uno en el primer registro pertenece a $\mathcal{B}$. Por otro lado, el programa con dos instrucciones donde cada una escribe cero en el primer registro tiene como código el número $\boldsymbol{\psi}(0,0) = \boldsymbol{\varphi_2}(1, \boldsymbol{\varphi_2}(0,0)) = 2$. Por lo tanto, dicho programa en la entrada 2, la cambia por 0, nuevamente escribe 0 y finaliza. De este modo, este programa nunca escribe el número 1. Por consiguiente, $\mathcal{B} \neq \mathbf{PROG}^A$. Por el teorema de Rice, One^A no es A-computable. Sin embargo, dado $e \in One^A$, preguntamos si $(e,e,1) \in Symb^A$. En caso afirmativo, $e \in One^A$, en otro caso $e \notin One^A$. Como $Symb^A$ es A-computable, One^A también debe ser A-computable, lo cual es una contradicción. Por lo tanto, $Symb^A$ no es A-computable.

(ii) Sea $\mathcal{B} := \{\mathbf{P}_e^A : \Phi_e(e) \downarrow 0\} \subseteq \mathbf{PROG}^A$. Notemos que el programa que devuelve 0 sin importar la entrada pertenece a $\mathcal{B}$. Y por otro lado, el programa que devuelve 1 sin importar la entrada no pertenece a $\mathcal{B}$. Por lo tanto, $\mathcal{B} \neq \emptyset, PROG^A$. Por el teorema de Rice concluimos que $DiagZ^A$ no es A-computable.

(iii) Supongamos que Z^A es A-computable. Dado que para cualquier $x \in \omega$, $x \in DiagZ^A \leftrightarrow (x,x) \in Z^A$, tenemos que $DiagZ^A$ es A-computable, lo cual contradice el inciso anterior. Por lo tanto, Z^A no es A-computable.

(iv) Similar a los incisos anteriores, basta observar que existen programas que se detienen en todas sus entradas como lo es aquel que devuelve 0 sin importar la entrada, y que existen programas que en algunas entradas no se detienen como lo es aquel que si tiene como entrada 0 escribe 0 en el primer registro y regresa a esta instrucción si el contenido del primer registro es 0 y en otro caso el programa termina. Esto nos permite concluir que T^A no es A-computable.

(v) La demostración es similar al inciso anterior.

(vi) Como en el inciso 4., observemos que el programa cuya única instrucción consiste en escribir 0 en el primer registro tiene código $\boldsymbol{\psi}(0) = 0$, devuelve 0 en cualquier entrada. Por otro lado notemos que el programa que escribe dos veces 0 en el primer registro tiene como código $\boldsymbol{\psi}(0,0) = 2$, sin embargo, dicho programa en ninguna entrada devuelve 2. Por el teorema de Rice, podemos concluir que E^A no es A-computable.

(vii) Veamos que existe un código de programa que pertenece a $Range^A$ y otro código que no. Para ello, consideremos el programa que devuelve el cuadrado mas uno de una entrada. Así, dado $n \in \omega$, para n tenemos que $\Phi^A_{e_0}(n) \downarrow$ $n^2 + 1 > n$, donde e_0 es el código del programa descrito. Por otro lado, basta considerar al programa que en cualquier entrada devuelve 0. El código de este programa no está en $Range^A$. Por el teorema de Rice, concluimos que $Range^A$ no es A-computable.

(viii) Dado que f es A-computable, existe un programa que devuelve los mismos valores de la función. Por otro lado, la función $f + 1$ es computable, por consiguiente existe un código de programa e que no pertenece a Eq^A_f. Por el teorema de Rice, concluimos que Eq^A_f no es A-computable.

(ix) Sea $e_0 \in \omega$ el código de un programa que se detenga en al menos una entrada. Consideremos el conjunto $S^A_{e_0} := \{e \in \omega : \forall x \in \omega\ (\Phi^A_e(x) \downarrow\leftrightarrow \Phi^A_{e_0}(x) \downarrow)\}$. Afirmamos que $S^A_{e_0}$ no es A-computable. En efecto, $e_0 \in S^A_{e_0}$ y además el programa que en la entrada en la que se detiene $\Phi^A_{e_0}$ no se detiene, no pertenece a $S^A_{e_0}$. Por el teorema de Rice, $S^A_{e_0}$ no es A-computable. De modo que si Sim^A fuera A-computable, entonces dado $e \in \omega$ siempre podríamos determinar de forma A-computable si $e \in S^A_{e_0}$ pregúntando si $(e, e_0) \in Sim^A$, lo cual haría que $S^A_{e_0}$ fuera A-computable, lo cual es falso. Por consiguiente, Sim^A no es A-computable.

□

Capítulo 3

Otros temas importantes en la Teoría de la Computabilidad

3.1. Computabilidad Enumerable

Otra utilidad de un programa consiste en «enumerar» los elementos de un conjunto. Esto es que dado un conjunto, exista un programa tal que todo elemento del conjunto pueda obtenerse a partir de dicho programa ejecutado en algún número natural como entrada. Una definición formal es la siguiente:

Definición 3.1.1. *(i)* Sean $n \geq 1$ y $X \subseteq \omega^n$, diremos que X es **computable enumerable** si y sólo si $X = \emptyset$ o existe una función computable $f : \omega \to \omega$ tal que im $f = \boldsymbol{\varphi_n}[X]$.

(ii) Sea $X \subseteq \omega^{<\mathbb{N}}$, diremos que X es **computable enumerable** si y sólo si $X = \emptyset$ o existe una función computable $f : \omega \to \omega$ tal que im $f = \boldsymbol{\psi}[X]$.

(iii) Sean $n \geq 1$ y $A \subseteq \omega$, un conjunto $X \subseteq \omega^n$ es A-**computable enumerable** (A-c.e.) si y sólo si $X = \emptyset$ o existe una función A-computable $f : \omega \to \omega$ tal que im $f = \boldsymbol{\varphi_n}[X]$.

(iv) Sea $A \subseteq \omega$, un conjunto $X \subseteq \omega^{<\mathbb{N}}$ es A-**computable enumerable** (A-c.e.) si y sólo si $X = \emptyset$ o existe una función A-computable $f : \omega \to \omega$ tal que im $f = \boldsymbol{\psi}[X]$.

De la definición anterior es evidente que $X \subseteq \omega^n$ para algún $n \geq 1$ es A-c.e. para algún $A \subseteq \omega$ si y sólo si $\boldsymbol{\varphi_n}[X]$ lo es. De la misma forma si $X \subseteq \omega^{<\mathbb{N}}$ entonces X es A-c.e. si y sólo si $\boldsymbol{\psi}[X]$ lo es. Por consiguiente, a partir de ahora, daremos definiciones y demostraremos teoremas sólo haciendo referencia a subconjuntos de ω.

Como una función es computable si y sólo si es $\emptyset$-computable, de ahora en adelante nos interesará mostrar resultados respecto a un oráculo arbitrario A, ya

que en particular son resultados sobre $\emptyset$.

Un enunciado equivalente a la definición anterior y que nos será de utilidad en muchos casos es el siguiente:

Proposición 3.1.1. *Dado $A \subseteq \omega$, un conjunto $X \subseteq \omega$ es A-computable enumerable si y sólo si existe una función $f : \omega \rightharpoonup \omega$ parcialmente A-computable tal que im $f = X$.*

Demostración. Supongamos que X es A-c.e. Si $X = \emptyset$ entonces definimos a $f : \omega \rightharpoonup \omega$ como la función cuyo dominio es el conjunto vacío. El programa que en cualquier entrada nunca se detiene es la función que coincide con f. Por lo tanto, ésta función es A-parcialmente computable y es tal que im $f = f[\emptyset] = \emptyset$.
Si $X \neq \emptyset$, entonces existe una función A-computable $f : \omega \to \omega$ tal que im $f = X$. En particular f es parcialmente A-computable y es tal que im $f = X$.

Ahora supongamos que existe una función $f : \omega \rightharpoonup \omega$ parcialmente A-computable tal que im $f = X$.

Si $X = \emptyset$, por definición es A-c.e.
Supongamos que $X \neq \emptyset$. Sea $a \in X$. Sea $e \in \omega$ tal que $\Phi_e^A = f$. Sea $g : \omega \to \omega$ la siguiente función:

$$g(x) = \begin{cases} f(z) & \text{si } z \in W_{e,t+1}^A \setminus W_{e,t}^A \text{ donde } \boldsymbol{\varphi_2}^{-1}(x) = (z,t) \\ a & \text{en otro caso} \end{cases}$$

Observemos que g es en efecto una función ya que $\boldsymbol{\varphi_2}^{-1}$ lo es y además si $z \in W_{e,t+1}$ para algún $t+1 \in \omega$ entonces $z \in W_e$, lo cual implica que $z \in$ dom Φ_e^A, es decir $z \in$ dom f.

Afirmamos que im $g = X$. Veamos que im $g \subseteq X$. Sea $y \in \omega$ tal que $y = g(x)$ para algún $x \in \omega$. Sean $z, t \in \omega$ tales que $\boldsymbol{\varphi_2}^{-1}(x) = (z,t)$. Si $z \in W_{e,t+1}^A \setminus W_{e,t}^A$ entonces $y = g(x) = f(z)$. Como im $f = X$, $y \in X$. Ahora bien si no ocurre que $z \in W_{e,t+1}^A \setminus W_{e,t}^A$ entonces $y = g(x) = a$. Como $a \in X$, $y \in X$. En cualquier caso tenemos que $y \in X$, por lo tanto im $g \subseteq X$. Ahora sea $y \in \omega$ tal que $y \in X$. Por lo tanto existe $z \in \omega$ tal que $z \in$ dom f y $f(z) = y$. Luego $z \in$ dom Φ_e^A, por lo tanto $z \in W_e^A$. Por consiguiente existe $t+1 \in \omega$ tal que $z \in W_{e,t+1}^A \setminus W_{e,t}^A$. Sea $x = \boldsymbol{\varphi_2}(z,t)$. Entonces $g(x) = f(z) = y$. Por lo tanto $y \in$ im g. Concluimos que im $g = X$.

Afirmamos que g es A-computable. Para computar los valores de g, dado x, obtenemos de forma computable valores z, t tales que $\boldsymbol{\varphi_2}^{-1}(x) = (z,t)$ verificamos si $z \in W_{e,t+1} \setminus W_{e,t}$ mediante la ejecución del programa $\mathbf{P}_e^A$ únicamente $t+1$ pasos y verificar si ya se devolvió el valor $z \in \omega$. En caso de que así sea entonces devolvemos el valor de $f(z)$ de forma A-computable, en caso contrario únicamente devolvemos

a. En cualquier caso devolvemos el valor de $g(x)$.

Concluimos que X es un conjunto A-computable enumerable.

□

Una forma de identificar a un conjunto A-c.e. y también a un conjunto A-computable, es la siguiente:

Proposición 3.1.2. *Dado $A \subseteq \omega$, $X \subseteq \omega$ es A-computable si y sólo si X y $\omega \setminus X$ son A-c.e.*

Demostración. Sean $A, X \subseteq \omega$.
Supongamos que X es A-computable. Si $X = \emptyset$, por definición es c.e. y por lo tanto A-c.e.

Si $X \neq \emptyset$, sea $\mathbf{P}_e^A$ el programa que hace a X un conjunto A-computable. La función parcial $f : \omega \nrightarrow \omega$ tal que dom $f = X$ y $\forall x \in$ dom $f(f(x) = x)$ es parcialmente A-computable. En efecto, definimos un programa tal que dado $x \in \omega$ preguntamos con el programa $\mathbf{P}_e^A$ si dicho elemento pertenece a X. En caso afirmativo devolvemos dicho número, en caso contrario el programa no se detiene. La función parcial inducida por este programa es tal que su imagen es X. Por lo tanto, X es A-c.e. La demostración es similar para $\omega \setminus X$.

Supongamos que tanto X como $\omega \setminus X$ son A-c.e. Sean $f, g : \omega \to \omega$ funciones A-computables tales que im $f = X$ y im $g = \omega \setminus X$. Consideremos el programa que en una entrada $x \in \omega$ busca el menor $m \in \omega$ tal que $f(m) = x$ ó $g(m) = x$. Si ocurre que $f(m) = x$ entonces el programa devuelve 1, si ocurre que $g(m) = x$ entonces el programa devuelve 0. La función inducida por este programa es la función característica de X. Por lo tanto, X es A-computable. □

Otra forma de caracterizar a los conjuntos A-c.e. es la siguiente:

Proposición 3.1.3. *Dados dos conjuntos $A, X \subseteq \omega$, X es A-c.e. si y sólo si existe un predicado A-computable $R(x, y)$ tal que $X = \{x \in \omega : \exists z \in \omega(R(x, z))\}$.*

Demostración. Sean $A, X \subseteq \omega$. Supongamos que X es A-c.e. Entonces existe una función A-computable $f : \omega \to \omega$ tal que im $f = X$. Sea $\mathbf{P}_e^A$ el programa que hace a f A-computable. Sea $R(x, z)$ el predicado que es verdadero si y sólo si $f(z) = x$. Observemos que R es A-computable. En efecto, dados $x, z \in \omega$, con el programa $\mathbf{P}_e^A$ calculamos el valor de dicho programa en z. Si el resultado es x, devolvemos 1 y en caso contrario devolvemos 0. Por lo tanto, R es A-computable. Afirmamos que $X = \{x \in \omega : \exists z \in \omega(R(x, z))\}$. Si $x \in X$ entonces existe $z \in \omega$ tal que $f(z) = x$, es decir tal que $R(x, z)$. Por otro lado si $x \in \omega$ es tal que existe $z \in \omega$ tal que $R(x, z)$ entonces $f(z) = x$ y por lo tanto $x \in X$. Concluimos que $X = \{x \in \omega : \exists z \in \omega(R(x, z))\}$.

Supongamos que existe un predicado A-computable $R(x,y)$ tal que $X = \{x \in \omega : \exists z \in \omega(R(x,z))\}$. Sea $f : \omega \rightharpoonup \omega$ la función parcial tal que $f(x) = x$ si y sólo si existe $y \in \omega$ tal que $R(x,y)$. Ésta función es parciamente A-computable por el programa que en un entrada x, busca el menor $m \in \omega$ tal que $R(x,m)$ y en caso de que lo encuentre el programa se detiene y devuelve a x, en caso contrario el programa busca indefinidamente dicho número. También im $f = \{x \in \omega : \exists z \in \omega(R(x,z))\} = X$. Lo cual nos permite concluir que X es A-c.e. □

Lo mencionado anteriormente nos da las herramientas para poder enunciar las siguientes equivalencias:

Teorema 3.1.1. *Dados dos conjuntos $A, X \subseteq \omega$, las siguientes son equivalentes:*

(i) *X es A-computable enumerable.*

(ii) *Existe una función parcialmente A-computable $f : \omega \rightharpoonup \omega$ tal que im $f = X$.*

(iii) *Existe un predicado A-computable $R(x,y)$ tal que*

$$X = \{x \in \omega : \exists y \in \omega(R(x,y))\}$$

(iv) *Existe $e \in \omega$ tal que $X = W_e^A$.*

(v) *X es el dominio de alguna función parcialmente A-computable.*

(vi) *Existe una función computable $f : \omega \to \omega$ tal que $x \in X$ si y sólo si $\boldsymbol{\varphi_2}^{-1}(f(x)) \in K^A$.*

Demostración. Sean $A, X \subseteq \omega$. Por los resultados anteriores, *(i)*, *(ii)* y *(iii)* son equivalente entre sí.

Veamos que *(iii)* implica *(iv)*. Sea $R(x,y)$ el predicado de la hipótesis. Sea $f : \omega \rightharpoonup \omega$ la función parcial tal que $f(x) = x$ si y sólo si existe $y \in \omega$ tal que $R(x,y)$. En resultados anteriores hemos visto que f es parcialmente A-computable y que im $f = X$. Sabemos que existe $e \in \omega$ tal que $\Phi_e^A = f$. Si $x \in \text{dom } \Phi_e^A$ entonces $x \in \text{dom } f$, lo que implica que $x = f(x) \in X$. Luego, dom $\Phi_e^A \subseteq X$. Por otro lado si $x \in X$, existe $y \in \text{dom } f$ tal que $f(y) = x$, como $f(y) = y$, tenemos que $x = y$. Por lo tanto $x \in \text{dom } f = \text{dom } \Phi_e^A$. Concluimos que $X = \text{dom } \Phi_e^A = W_e^A$.

Veamos que *(iv)* implica *(v)*. Si $X = \text{dom } \Phi_e^A = W_e^A$ para algún $e \in \omega$. Como Φ_e^A es parcialmente A-computable el resultado se sigue.

Veamos que *(v)* implica *(vi)*. Supongamos que $X = \text{dom } g$ para alguna función g parcialmente A-computable. Entonces existe $e \in \omega$ tal que $\Phi_e^A = g$. Por lo que

$X = \text{dom } \Phi_e^A$. Sea $f : \omega \to \omega$ tal que $f(x) = \boldsymbol{\varphi_2}(e, x)$. Es evidente que f es computable. Además que,

$$\begin{aligned} x \in X &\Leftrightarrow \Phi_e^A(x)\downarrow \\ &\Leftrightarrow (e,x) \in K^A \\ &\Leftrightarrow \boldsymbol{\varphi_2}^{-1}(\boldsymbol{\varphi_2}(e,x)) \in K^A \\ &\Leftrightarrow \boldsymbol{\varphi_2}^{-1}(f(x)) \in K^A \end{aligned}$$

Para ver que *(vi)* implica *(ii)*, sea f la función de la hipótesis. Definimos $g : \omega \rightharpoonup \omega$ la función parcial tal que

$$g(x) = \begin{cases} x & \text{si } \boldsymbol{\varphi_2}^{-1}(f(x)) \in K^A \\ indefinida & \text{si ocurre otro caso} \end{cases}$$

.

Notemos que g es parcialmente A-computable ya que el programa que en una entrada x, calcula $f(x)$ y posteriormente $\boldsymbol{\varphi_2}^{-1}(f(x))$ y luego verifica si la pareja obtenida pertenece a K^A mediante la ejecución del programa que hace a Ψ_U^A es parcialmente A-computable. Si éste se detiene entonces el programa devuelve x, en caso contrario el programa no se detendrá porque Ψ_U^A no se detuvo en el par obtenido. Afirmamos que $X = \text{im } g$. En efecto,

$$\begin{aligned} x \in X &\Leftrightarrow \boldsymbol{\varphi_2}^{-1}(f(x)) \in K^A \\ &\Leftrightarrow x \in \text{dom } g \\ &\Leftrightarrow x = g(x) \in \text{im } g \end{aligned}$$

Por lo tanto, $X = \text{im } g$.

□

Usando el resultado anterior nos será más sencillo demostrar cuáles conjuntos sí son A-c.e.

Proposición 3.1.4. *Para cualquier $A \subseteq \omega$, los siguientes conjuntos son A-c.e.*

(i) $A' = \{x \in \omega : \Phi_x^A(x)\downarrow\}$

(ii) $K^A = \{(e,x) \in \omega \times \omega : \Phi_e^A(x)\downarrow\}$

(iii) $Symb^A = \{(e,x,s) \in \omega^3 : \exists t, r \in \omega\ (exp(con^A(e,x,t), r) \dot{-} 1 = s)\}$

(iv) $DiagZ^A = \{x \in \omega : \Phi_x^A(x)\downarrow 0\}$

(v) $E^A = \{x \in \omega : \exists y \in \omega\ (\Phi_x^A(y)\downarrow x)\}$

Demostración. *(i)* Sea $f : \omega \to \omega$ la función tal que $f(x) = \boldsymbol{\varphi_2}(x,x)$. Ésta función es computable y es tal que $x \in A'$ si y sólo si $f(x) \in K^A$. Por lo tanto, A' es A-c.e.

(ii) Es inmediato del hecho de que la función identidad es computable.

(iii) Sea $f : \omega \rightharpoonup \omega$ la función parcial tal que $f(x) = x$ si $\boldsymbol{\varphi}_{\mathbf{3}}^{-1}(x) \in Symb^A$ y en otro caso permanece indefinida. Notemos que ésta función es A-parcialmente computable ya que dado x, calculamos $\boldsymbol{\varphi}_{\mathbf{3}}^{-1}(x)$, digamos (e, x, s), y buscamos el primer $m \in \omega$ tal que $\boldsymbol{\varphi}_{\mathbf{2}}^{-1}(m)$, digamos (t, r) y verificamos de forma A-computable si $exp(con^A(e, x, t), r)\dot{-}1 = s$, en caso de que se encuentre entonces devolvemos x, en caso contrario el programa corre indefinidamente. Afirmamos que im $f = \boldsymbol{\varphi}_{\mathbf{3}}[Symb^A]$. En efecto, si $y \in \omega$ es tal que $f(x) = y$ con $x \in$ dom f entonces $y = x$ y por lo tanto $y \in \boldsymbol{\varphi}_{\mathbf{3}}[Symb^A]$. Y viceversa si $x \in \boldsymbol{\varphi}_{\mathbf{3}}[Symb^A]$ entonces $\boldsymbol{\varphi}_{\mathbf{3}}^{-1}(x) \in Symb^A$ y por lo tanto $x \in$ dom f, lo que implica que $x = f(x) \in$ im f. Con esto concluimos que im $f = \boldsymbol{\varphi}_{\mathbf{3}}[Symb^A]$. Luego, $Symb^A$ es A-c.e.

(iv) Sea $f : \omega \rightharpoonup \omega$ la función parcial tal que $f(x) = x$ si $\Phi_x^A(x) \downarrow 0$ y está indefinida en otro caso. Dicha función es A-parcialmente computable ya que basta verificar si el programa x en la entrada x se detiene y devuelve 0, en caso de que no se detenga, el programa principal tampoco se detiene. Notemos que im $f = DiagZ^A$. En efecto, $f(x) = x$ con $x \in$ dom f si y sólo si $\Phi_x^A(x) \downarrow 0$, lo cual es equivalente a que $x \in DiagZ^A$. Concluimos que $DiagZ^A$ es A-c.e.

(v) Sea $f : \omega \rightharpoonup \omega$ la función parcial tal que $f(x) = x$ si $\exists y \in \omega\ (\Phi_x^A(y) \downarrow x)$. Afirmamos que f es A-parcialmente computable. En efecto, dado x, buscamos el menor $m \in \omega$ tal que $\Phi_x^A(m) \downarrow x$ de la siguiente forma. El programa ejecuta un paso con 0 como entrada y si observa que se detiene y su resultado es x entonces se devuelve x, en caso contrario se dispone a correr el programa con dos pasos en 0, si no se detiene ahora corre el programa un paso con 1. Si aún no se detiene se ejecutan tres pasos con 0 como entrada, si no entonces se ejecutan dos pasos con 1, si no entonces un paso con 2 y así sucesivamente. Notemos que im $f = E^A$. En efecto, $x = f(x)$ con $x \in$ dom f si y sólo si $\exists y \in \omega\ (\Phi_x^A(y) \downarrow x)$, lo cual es equivalente a que $x \in E^A$. Por lo tanto, im $f = E^A$. Lo cual implica que E^A es A-c.e.

□

Observemos que los conjuntos de la proposición anterior no son A-computables, por lo que son ejemplos de conjuntos que son A-computables enumerables pero que no son A-computables. Esto en contraste con el hecho de que todo conjunto A-computable es A-computable enumerable mencionado en la proposición 3,1,2.

Ahora mostraremos un par de conjuntos que no son A-c.e.

Teorema 3.1.2. *Sea $A \subseteq \omega$. Los siguientes conjuntos no son A-computables enumerables.*

(i) $\omega \setminus A'$

(ii) $T^A = \{e \in \omega : \forall x \in \omega\ (\Phi_e^A(x) \downarrow)\}$

Demostración. Sea $A \subseteq \omega$.

(i) Supongamos que $\omega \setminus A'$ es A-c.e. Por la proposición 3,1,4 sabemos que A' es A-c.e. Por lo tanto por la proposición 3,1,2, A' es A-computable, lo cual es una contradicción. Por lo tanto $\omega \setminus A'$ no es A-c.e.

(ii) Supongamos que T^A es A-c.e. Sea $f : \omega \to \omega$ una función A-computable tal que im $f = T^A$. Sea $g : \omega \to \omega$ tal que $g(x) = \Phi^A_{f(x)}(x) + 1$. Notemos que dom $g = \omega$. Por lo tanto, existe $n_0 \in \omega$ tal que $g(x) = \Phi^A_{f(n_0)}(x)$. Entonces, $\Phi^A_{f(n_0)}(n_0) + 1 = g(n_0) = \Phi^A_{f(n_0)}(n_0)$, lo cual es una contradicción. Concluimos que T^A no es A-c.e.

□

3.2. Reducibilidades

En este apartado mencionaremos algunos tipos de formas de comparar la complejidad de conjuntos de números naturales.

3.2.1. Reducibilidad m

Definición 3.2.1. Sean $A, B \subseteq \omega$.

(i) Diremos que A es **m-reducible** a B lo cual denotamos por $A \leq_m B$ si y sólo si existe una función computable $f : \omega \to \omega$ tal que $x \in A$ si y sólo si $f(x) \in B$.

(ii) Diremos que A es **m-equivalente** a B lo cual denotamos por $A \equiv_m B$ si y sólo si $A \leq_m B$ y $B \leq_m A$.

Observación. La relación $\equiv_m$ es una relación de equivalencia.

Demostración. *(i)* Veamos que $\equiv_m$ es reflexiva. Sea $A \subseteq \omega$. Sea $i : \omega \to \omega$ la función identidad de los números naturales. Esta función es computable y es tal que $x \in A$ si y sólo si $i(x) = x \in A$. Por lo tanto, $A \leq_m A$, lo cual nos implica que $A \equiv_m A$.

(ii) Veamos que $\equiv_m$ es simétrica. Sean $A, B \subseteq \omega$ tales que $A \equiv_m B$. Entonces $A \leq_m B$ y $B_m \leq_m A$, lo cual implica que $B \equiv_m A$.

(ii) Veamos que $\equiv_m$ es transitiva. Sean $A, B, C \subseteq \omega$, tales que $A \equiv B$ y $B \equiv_m C$. Entonces existen $f, g : \omega \to \omega$ funciones computables tales que $x \in A$ si y sólo si $f(x) \in B$ y $y \in B$ si y sólo si $g(y) \in C$. Se cumple entonces que $g \circ f : \omega \to \omega$ es computable y que $x \in A$ si y sólo si $f(x) \in B$ si y sólo si $(g \circ f)(x) = g(f(x)) \in C$. Por lo tanto, $A \leq_m C$. La demostración de $C \leq_m A$ es similar. Por lo tanto, $A \equiv_m C$.

Por lo tanto $\equiv_m$ es una relación de equivalencia. □

Un ejemplo importantes de conjuntos que son m-equivalentes es el siguiente.

Proposición 3.2.1. *Para cualquier conjunto $A \subseteq \omega$, $A' \equiv_m \boldsymbol{\varphi_2}[K^A]$.*

Demostración. Sea $A \subseteq \omega$. Como A' es A-c.e. por el teorema 3,1,1 tenemos que $A' \leq_m \boldsymbol{\varphi_2}[K^A]$.

Afirmamos que $\boldsymbol{\varphi_2}[K^A] \leq_m A'$. Sea $\mathbf{P}^A_{e_0}$ el programa tal que

$$\Phi^A_{e_0}(e,x,y) = \begin{cases} 1 & \text{si } \Phi^A_e(x)\downarrow \\ \text{indefinida} & \text{si ocurre otro caso} \end{cases}$$

y tal que $\mathbf{P}^A_{e_0}$ tiene las instrucciones de Ψ^A_U y un último par de instrucciones que escriben en el primer registro el número 1. Por consiguiente podemos obtener a e_0 a partir del código de Ψ^A_U y de las dos instrucciones mencionadas.

Ahora bien, por el teorema 2,2,2, existe una función computable e inyectiva $s : \omega^3 \to \omega$ tal que $\Phi^A_{e_0}(e,x,y) = \Phi_{s(e_0,e,x)}(y)$. Definimos $g : \omega \times \omega \to \omega$ la función tal que $g(e,x) = s(e_0,e,x)$. Tenemos que g es computable e inyectiva. Definimos $f : \omega \to \omega$ la función tal que $f(x) = g(\boldsymbol{\varphi_2}^{-1}(x))$. Es evidente que f es computable. Afirmamos que $x \in \boldsymbol{\varphi_2}[K^A]$ si y sólo si $f(x) \in A'$. Observemos que

$$\begin{aligned} x \in \boldsymbol{\varphi_2}[K^A] &\Leftrightarrow (e,z) = \boldsymbol{\varphi_2}^{-1}(x) \in K^A \\ &\Leftrightarrow \Phi^A_e(z)\downarrow \\ &\Leftrightarrow \forall y \in \omega(\Phi^A_{e_0}(e,z,y) = 1) \\ &\Leftrightarrow \forall y \in \omega(\Phi^A_{s(e_0,e,z)}(y) = 1) \\ &\Leftrightarrow \Phi^A_{s(e_0,e,z)}(s(e_0,e,z)) = 1 \\ &\Leftrightarrow \Phi^A_{s(e_0,e,z)}(s(e_0,e,z))\downarrow \\ &\Leftrightarrow g(e,z) = s(e_0,e,z) \in A' \\ &\Leftrightarrow f(x) = g(\boldsymbol{\varphi_2}^{-1}(x)) \in A' \end{aligned}$$

Concluimos que $x \in \boldsymbol{\varphi_2}[K^A]$ si y sólo si $f(x) \in A'$. Por lo tanto, $\boldsymbol{\varphi_2}[K^A] \leq_m A'$. Finalmente tenemos que $A' \equiv_m \boldsymbol{\varphi_2}[K^A]$. □

Corolario 3.2.1. *Un conjunto $X \subseteq \omega$ es computable enumerable si y sólo si $X \leq_m \emptyset'$.*

Demostración. Que X sea c.e. equivale a que X sea $\emptyset$-c.e. Por el teorema 3,1,1 esto equivale a que $X \leq_m \boldsymbol{\varphi_2}[K^\emptyset]$. Por la proposición anterior, esto es equivalente a que $X \leq_m \emptyset'$. □

3.2.2. Reducibilidad Turing

Definición 3.2.2. Sean $A, B \subseteq \omega$.

(i) Diremos que A es **Turing-reducible** a B lo cual denotamos por $A \leq_T B$ si y sólo si A es B-computable.

(ii) Diremos que A es **Turing-equivalente** a B, lo cual denotamos por $A \equiv_T B$ si y sólo si $A \leq_T B$ y $B \leq_T A$.

Observación. La relación de Turing-equivalencia sobre conjuntos de números naturales es una relación de equivalencia.

Demostración. *(i)* Veamos que $\equiv_T$ es reflexiva. Sea $A \subseteq \omega$. Por el ejemplo 1,1,9 tenemos que A es A-computable, se sigue que $A \equiv_T A$.

(ii) Veamos que $\equiv_T$ es simétrica. Sean $A, B \subseteq \omega$ tal que $A \equiv_T B$, entonces A es B-computable y B es A-computable. Por lo tanto, $B \equiv_T A$.

(iii) Veamos que $\equiv_T$ es transitiva. Sean $A, B, C \subseteq \omega$ tal que $A \equiv_T B$ y $B \equiv_T C$. Veamos que $A \equiv_T C$, es decir que $A \leq_T C$ y $C \leq_T A$. Dado que A es B-computable y B es C-computable tenemos que existen programas $\mathbf{P}^B, \mathbf{Q}^C$ tales que $\chi_A = \Phi_{\mathbf{P}^B}$ y $\chi_B = \Phi_{\mathbf{Q}^C}$. Sea $\mathbf{R}$ el programa que toda instrucción O_k^B de $\mathbf{P}^B$ es sustituida por las instrucciones del programa $\mathbf{Q}^C$ de tal forma que su entrada sea el registro r_k y el resultado lo coloque en el mismo registro sin afectar el contenido de los demás registros. Es evidente que $\chi_A = \mathbf{R}^C$. Por lo cual concluimos que A es C-computable. La prueba para ver que C es A-computable es similar. Por lo tanto, podemos concluir que $A \equiv_T C$.

□

Definición 3.2.3. Las clases de equivalencia definidas por la relación $\equiv_T$ se denominan **grados de Turing**.

Observación. Para cualquier conjunto $A \subseteq \omega$, $\emptyset \leq_T A$.

Demostración. Como $\emptyset$ es $\emptyset$-computable entonces $\emptyset$ es computable. Por lo tanto, $\emptyset$ es A-computable, es decir $\emptyset \leq_T A$. □

Observación. Sea $A \subseteq \omega$, entonces A es computable si y sólo si $A \equiv_T \emptyset$.

Demostración. Recordemos que A es computable implica que A es $\emptyset$-computable, por lo tanto $A \leq_T \emptyset$. Por la observación anterior, $\emptyset \leq_T A$. Concluimos que $A \equiv_T \emptyset$. □

Proposición 3.2.2. *Sean $A, B \subseteq \omega$. Si $A \leq_m B$ entonces $A \leq_T B$.*

Demostración. Sean $A, B \subseteq \omega$ tales que $A \leq_m B$. Sea $f : \omega \to \omega$ función computable tal que $x \in A$ si y sólo si $f(x) \in B$. Sea $\mathbf{Q}$ el programa que hace a f computable. Definimos $\mathbf{P}^B$ como el programa que resulta de la concatenación de $\mathbf{Q}$ y el programa cuya única instrucción es O_1^B. En una entrada x, el programa $\mathbf{P}$ obtiene el valor de $f(x)$ y pregunta si dicho valor pertenece a B. Si $x \in A$ entonces $f(x) \in B$ por lo que el programa devuelve 1, es decir devuelve $\chi_A(x)$. Si $x \notin A$ entonces $f(x) \notin B$, el programa devuelve 0, es decir devuelve $\chi_A(x)$. Concluimos que $\chi_A = \Phi_{\mathbf{P}^B}$, por lo tanto A es B-computable, es decir $A \leq_T B$. □

Observemos que el recíproco de la proposición anterior no se cumple. Para notar esto tomemos $A = \omega \setminus \emptyset'$ y a $B = \emptyset'$. Para obtener que $\omega \setminus \emptyset' \leq_T \emptyset'$, consideremos el programa en una entrada $x \in \omega$, pregunta si $x \in \emptyset'$, en caso de que si el programa devuelve 0 y en caso contrario devuelve 1. Este programa utiliza a $\emptyset'$ como oráculo y su función inducida es la función característica de $\omega \setminus \emptyset'$. Por lo tanto $\omega \setminus \emptyset'$ es $\emptyset'$-computable, es decir $\omega \setminus \emptyset' \leq_T \emptyset'$.

Por otro lado por el teorema 3,1,2 sabemos que $\omega \setminus \emptyset'$ no es computable enumerable, lo cual por el corolario 3,2,1 equivale a que no ocurra que $\omega \setminus \emptyset' \leq_m \emptyset'$.

Por lo mencionado previamente también podemos decir que existen conjuntos $\leq_T$ que $\emptyset'$ que no son computables enumerables.

Una relación importante entre los conjuntos A' y $\boldsymbol{\varphi_2}[K^A]$ es la siguiente.

Proposición 3.2.3. $A' \equiv_T \boldsymbol{\varphi_2}[K^A]$.

Demostración. Se sigue de $A' \equiv_m \boldsymbol{\varphi_2}[K^A]$ y la proposición 3,2,2. □

Por lo dicho, identificaremos a los conjuntos A' y $\boldsymbol{\varphi_2}[K^A]$ como si fueran iguales, los denomiranemos **conjunto de parada relativo a** A y los denotaremos por A'. Para el conjunto de parada relativo a $\emptyset$ simplemente diremos **conjunto de parada**.

Proposición 3.2.4. *Para cualquier conjunto $A \subseteq \omega$ se cumple que $A \leq_T A'$ pero no que $A' \leq_T A$.*

Demostración. Como A es A-c.e. entonces el teorema 3,1,1 nos implica que $A \leq_m \boldsymbol{\varphi_2}[K^A]$. Por la proposición 3,2,2, $A \leq_T \boldsymbol{\varphi_2}[K^A]$. Como $\boldsymbol{\varphi_2}[K^A] \leq_T A'$ tenemos que $A \leq_T A'$. Sabemos que A' no es A-computable, por lo tanto no ocurre que $A' \leq_T A$. □

Dados $A, B \subseteq \omega$ si ocurre que $A \leq_T B$ pero no $B \leq_T A$ entonces diremos que $A <_T B$. Notemos que esta relación no es reflexiva ni simétrica sino sólo transitiva.

Corolario 3.2.2. *Para todo $A \subseteq \omega$, $A <_T A'$.*

Demostración. Es la proposición anterior. □

A partir de ahora dado un conjunto $A \subseteq \omega$, diremos que A' es el **salto de Turing** de A. Denotamos por A'' a la doble iteración y por $A^{(n)}$ a la n-ésima iteración del salto.

Para el conjunto $\emptyset$, denotamos por **0** a su grado de Turing, por **0'** al grado de $\emptyset'$ y por $\mathbf{0^{(n)}}$ al grado de $\emptyset^{(n)}$.

Para denotar al grado de Turing de un conjunto A usaremos la letra **a**, para denotar el grado del salto de A escribimos **a'** y de manera similar para los demás saltos y demás letras del alfabeto.

Diremos también que un grado de Turing **a** es un grado de Turing computable enumerable o recursivo enumerable si y sólo si algún representante de **a** es un conjunto computable enumerable.

A partir del orden $\leq_T$ podemos inducir un orden sobre los grados de Turing como sigue.

Definición 3.2.4. Sean **a** y **b** dos grados de Turing.

(i) Diremos que $\mathbf{a} \leq \mathbf{b}$ si y sólo si $A \leq_T B$ para algunos conjuntos A, B en los grados **a** y **b** respectivamente.

(ii) Diremos que $\mathbf{a} < \mathbf{b}$ si y sólo si $\mathbf{a} \leq \mathbf{b}$ pero no ocurre que $\mathbf{b} \leq \mathbf{a}$.

Notemos que $\leq$ de la definición anterior no depende de que conjuntos A y B se elijan. En efecto si A, B, C, D son conjuntos tales que A, C están en el grado **a** y B, D en **b** entonces $A \equiv_T C$ y $B \equiv_T D$. Si $A \leq_T B$ entonces $C \leq_T B$, por la simetría y transitividad de $\equiv_T$. Nuevamente por la transitividad de $\equiv_T$ tenemos que $C \leq_T D$. Concluimos que el orden anterior no depende de los representantes de cada grado de Turing. De manera similar se puede verificar para $<$.

Una observación sobre $\leq$ y $<$ es la siguiente.

Proposición 3.2.5. *(a) Para cualesquiera grados de Turing* **a** *y* **b** *se cumple lo siguiente.*

(i) $\mathbf{a} \leq \mathbf{a}$.

(ii) Si $\mathbf{a} \leq \mathbf{b}$ *y* $\mathbf{b} \leq \mathbf{a}$ *entonces* $\mathbf{a} = \mathbf{b}$.

(iii) Si $\mathbf{a} \leq \mathbf{b}$ *y* $\mathbf{b} \leq \mathbf{c}$ *entonces* $\mathbf{a} \leq \mathbf{c}$.

(b) Para cualesquiera grados de Turing **a** *y* **b** *se cumple lo siguiente.*

(i) No ocurre que $\mathbf{a} < \mathbf{a}$.

(ii) Si $\mathbf{a} < \mathbf{b}$ *entonces no puede ocurrir que* $\mathbf{b} < \mathbf{a}$.

(iii) Si $\mathbf{a} < \mathbf{b}$ *y* $\mathbf{b} < \mathbf{c}$ *entonces* $\mathbf{a} < \mathbf{c}$.

Demostración. La demostración se sigue esencialmente de las propiedades de $\leq_T$ y de la definición de $<_T$. □

Un conjunto que satisface que para cualesquiera de sus elementos se cumplen los incisos *(i)*, *(ii)* y *(iii)* del inciso *(a)* se denomina **orden parcial**.

Y si satisface los incisos *(i)*, *(ii)* y *(iii)* del inciso *(b)* se denomina **orden parcial estricto**.

Por lo tanto el conjunto de grados de Turing es un órden parcial con el orden $\leq$ y es un orden parcial estricto con el orden $<$.

A continuación mencionamos algunos resultados interesantes sobre el conjunto de grados de Turing junto con el orden $\leq$ y $<$. Las demostraciones de estos resultados están fuera del objetivo en este trabajo pero el lector puede estudiar más del tema y revisarlas en [Soa].

Teorema 3.2.1. *(i) (Teorema de Friedberg-Muchnik, [Soa], Corolario 1.2. pág. 111) Existe un grado de Turing computable enumerable* $\mathbf{a}$ *tal que* $\mathbf{0} < \mathbf{a} < \mathbf{0}'$.

(ii) (Teorema de Densidad de Sacks, [Soa], Teorema 4.1. pág. 142) Para cualesquiera grados de Turing computables enumerables $\mathbf{a}$ *y* $\mathbf{b}$ *existe un grado de Turing computable enumerable* $\mathbf{c}$ *tal que* $\mathbf{a} < \mathbf{c} < \mathbf{b}$.

(iii) (Criterio de Completitud de Friedberg. [Soa], Teorema 3.1. pág. 97) Para cualquier grado de Turing $\mathbf{b}$ *tal que* $\mathbf{0}' \leq \mathbf{b}$ *existe un grado de Turing* $\mathbf{a}$ *tal que* $\mathbf{a}' = \mathbf{b}$.

(iv) (Existencia de grados minimales. [Soa], Teorema 5.7. pág. 106) Existe un grado de Turing $\mathbf{a}$ *tal que* $\mathbf{0} < \mathbf{a} < \mathbf{0}'$ *y tal que no existe ningún grado de Turing* $\mathbf{b}$ *tal que* $\mathbf{0} < \mathbf{b} < \mathbf{a}$. *Es decir, existe un grado de Turing minimal entre* $\mathbf{0}$ *y* $\mathbf{0}'$.

(v) (Existencia de $2^{\aleph_0}$ *grados de Turing incomparables. [Soa], Ejercicio 1.8. pág. 95) Existe un conjunto* I *de grados de Turing tal que* $|I| = 2^{\aleph_0}$ *y para cualesquiera* $\mathbf{a}, \mathbf{b} \in I$ *distintos no ocurre que* $\mathbf{a} \leq \mathbf{b}$ *ni* $\mathbf{b} \leq \mathbf{a}$. *Es decir, existen* $2^{\aleph_0}$ *grados de Turing incomparables entre sí dos a dos.*

En la figura 3,1 damos una idea gráfica del conjunto de grados de Turing.

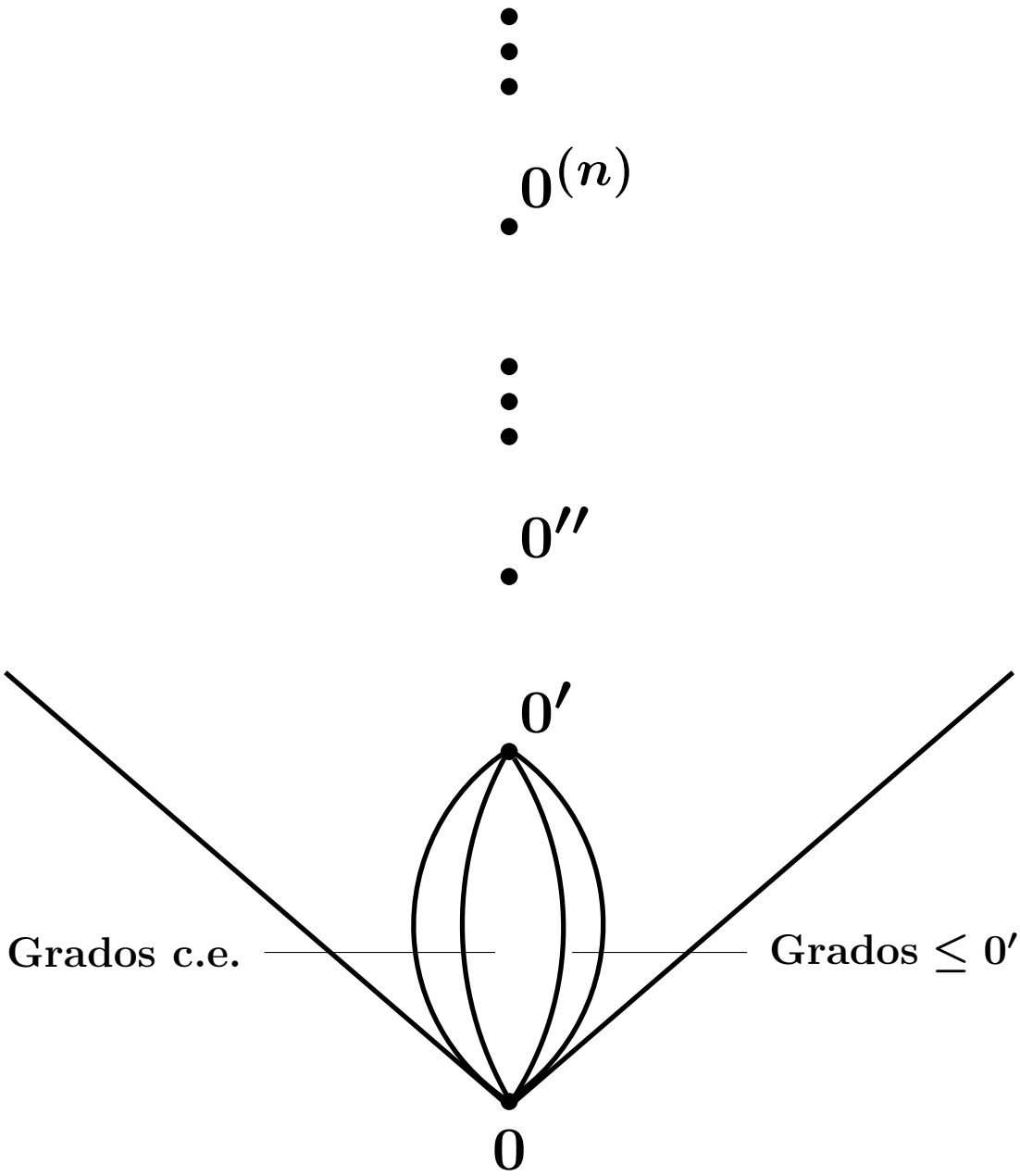

Figura 3.1: Grados de Turing

Capítulo 4

Estructuras desde el punto de vista de la Teoría de la Computabilidad

4.1. Preliminares de la Teoría de Estructuras/Modelos

En esta sección introduciremos la noción de lenguaje predicativo y estructura, con los cuales a grandes rasgos podemos representar sintácticamente una gran variedad de objetos matemáticos, tales como la noción de grupo, espacio vectorial, anillos, entre otros.
Comenzaremos definiendo lo que significa una signatura.

Definición 4.1.1. Una **signatura** Σ es una tupla que consiste de los siguientes conjuntos:

(i) $\{c_n : n \in I_C\}$ donde I_C es un segmento inicial, posiblemente vacío, de ω y cuyos elementos llamaremos *símbolos constantes de* Σ, cuando esté clara la signatura únicamente diremos *símbolos constantes*;

(ii) $\{R_i^{a_R(i)} : i \in I_R\}$ donde I_R es un segmento inicial de ω, posiblemente vacío, y $a_R : I_R \to \mathbb{N}$ es una función tal que $a_R(i)$ es la cantidad de parámetros, que también llamaremos **aridad**, de R_i. A los elementos de este conjunto los llamaremos *símbolos relacionales de* Σ, cuando esté clara la signatura únicamente diremos *símbolos relacionales*;

(iii) $\{f_i^{a_F(i)} : i \in I_F\}$ donde I_F es un segmento inicial de ω, posiblemente vacío, y $a_F : I_F \to \mathbb{N}$ es una función tal que $a_F(i)$ es la cantidad de parámetros, que también llamaremos **aridad**, de f_i. A los elementos de este conjunto los llamaremos *símbolos funcionales de* Σ, cuando esté clara la signatura únicamente diremos *símbolos funcionales*.

Diremos que la signatura Σ es **computable** si y sólo si las funciones a_R y a_F son parcialmente computables.

Definición 4.1.2. Sean Σ_1 y Σ_2 dos signaturas. Diremos que Σ_1 **es extendida por** Σ_2 si y sólo si todo símbolo constante, relacional y funcional de Σ_1 es un símbolo constante, relacional y funcional de Σ_2 respectivamente.

Observación. Si Σ es una signatura donde los segmentos iniciales I_R e I_F son finitos entonces Σ es computable. Es decir, si hay una cantidad finita de símbolos relacionales y funcionales, la signatura es computable.

Ahora bien, de modo que podamos construir proposiciones cuantificadas con los símbolos de una signatura.

Definición 4.1.3. Dada una signatura Σ, un **lenguaje predicativo** $\mathcal{L}$ es una tupla $(\mathcal{S}, \Sigma)$ donde $\mathcal{S}$ consiste de los siguientes conjuntos:

(i) $\{x_n : n \in \omega\}$ cuyos elementos llamaremos *símbolos variables*;

(ii) $\{\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow\}$ de *conectivos lógicos*;

(iii) $\{(,), , \}$ de *paréntesis* y una *coma* respectivamente;

(iv) $\{\forall, \exists\}$ de *símbolos cuantificadores*.

Además de ahora en adelante, para referirnos a un símbolo variable/constante únicamente diremos que es una variable/constante.
Observemos que la única diferencia entre dos lenguajes predicativos son sus signaturas.

Definición 4.1.4. Dado un lenguaje predicativo $\mathcal{L} = (\mathcal{S}, \Sigma)$, definimos su conjunto de términos, denotado por $\mathcal{L}$-**TERM** y satisface que:

(i) Toda variable está en $\mathcal{L}$-**TERM**.

(ii) Si Σ tiene constantes. Toda constante de Σ está en $\mathcal{L}$-**TERM**.

(iii) Si Σ tiene símbolos funcionales. Dados $t_1, ..., t_j$ en $\mathcal{L}$-**TERM** y f_i^j un símbolo funcional de aridad j de Σ, $f_i^j(t_1, ..., t_j)$ está en $\mathcal{L}$-**TERM**.

(iv) Si T es un conjunto y cumple los incisos *(i)*, *(ii)* y *(iii)* entonces $\mathcal{L}$-**TERM** $\subseteq T$.

A los elementos de $\mathcal{L}$-**TERM** les llamaremos $\mathcal{L}$-**términos**. Cuando el lenguaje sea claro, en las definiciones anteriores cambiamos $\mathcal{L}$-**TERM** por **TERM**.

Tenemos la siguiente observación.

Observación. Para un lenguaje predicativo $\mathcal{L} = (\mathcal{S}, \Sigma)$, t es un $\mathcal{L}$-término si y sólo si ocurre que t es igual a:

(i) una variable de Σ, x ó

(ii) una constante de Σ, c ó

(iii) un $\mathcal{L}$-término $f_i^j(t_1, ..., t_j)$ para algún funcional f_i^j de Σ y $t_1, ..., t_j$ $\mathcal{L}$-términos.

Demostración. Se sigue de la minimalidad del conjunto de $\mathcal{L}$-términos. □

Definición 4.1.5. Dado un lenguaje predicativo $\mathcal{L}$, definimos su conjunto de fórmulas bien formadas finitas, denotado por $\mathcal{L}$-**FORM**, como el conjunto tal que:

(i) Dados $t_1, ..., t_j$ en $\mathcal{L}$-**TERM** y R_i^j un símbolo relacional de aridad j, $R_i^j(t_1, ..., t_j)$ está en $\mathcal{L}$-**FORM**. A estas fórmulas las llamaremos **$\mathcal{L}$-fórmulas atómicas**.

(ii) Dada φ en $\mathcal{L}$-**FORM**, $\neg\varphi$ está en $\mathcal{L}$-**FORM**. A las $\mathcal{L}$-fórmulas atómicas y a las negaciones de $\mathcal{L}$-fórmulas atómicas las llamaremos **$\mathcal{L}$-fórmulas literales**.

(iii) Dadas φ, ψ en $\mathcal{L}$-**FORM**, $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \Rightarrow \psi), (\varphi \Leftrightarrow \psi)$ están en $\mathcal{L}$-**FORM**.

(iv) Dada una variable x_j y φ en $\mathcal{L}$-**FORM**, $(\forall x_j(\varphi))$ y $(\exists x_j(\varphi))$ están en $\mathcal{L}$-**FORM**.

(v) Si F es un conjunto y cumple los incisos *(i)*, *(ii)*, *(iii)* y *(iv)* entonces $\mathcal{L}$-***FORM*** $\subseteq F$.

A los elementos de $\mathcal{L}$-**FORM** les llamaremos **$\mathcal{L}$-fórmulas bien formadas finitas** ó **$\mathcal{L}$-fórmulas elementales finitas**. Cuando el lenguaje sea claro, en las definiciones anteriores cambiamos $\mathcal{L}$-**FORM** por **FORM**.

En ocasiones omitiremos los paréntesis de una fórmula bien formada finita a fin de que sea más comprensible.

Observación. Para un lenguaje predicativo $\mathcal{L} = (\mathcal{S}, \Sigma)$, φ es una $\mathcal{L}$-fórmula bien formada si y sólo si ocurre que φ es de alguna de las siguientes formas:

(i) $R_i^j(t_1, ..., t_j)$ para algún símbolo relacional R_i^j y algunos términos $t_1, ..., t_j$ ó

(ii) $\neg\psi$ para alguna fórmula ψ ó

(iii) $\psi_1 \square \psi_2$ para algunas fórmulas ψ_1, ψ_2 y $\square \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$ ó

(iv) $(\forall x(\psi))$ ó $(\exists x(\psi))$ para alguna variable x y fórmula ψ.

Demostración. Se sigue de la minimalidad del conjunto de $\mathcal{L}$-fórmulas bien formadas. □

Definición 4.1.6. Para cada término $t \in$ **TERM** definimos su conjunto de variables V como sigue:

(i) Si x es una variable, $V(x) = \{x\}$

(ii) Si c es una constante, $V(c) = \emptyset$

(iii) Si f_i^j es un símbolo funcional y $t_1, ..., t_j$ son términos, $V(f_i^j(t_1, ..., t_j)) = \bigcup_{k=1}^{j} V(t_k)$.

También nos interesa especificar la variables de una fórmula bien formada.

Definición 4.1.7. Definimos el conjunto de **variables libres** de una fórmula φ, el cual denotaremos por $FV(\varphi)$, como sigue:

(i) Si R_i^j es un símbolo relacional y $t_1, ..., t_j$ son términos, $FV(R_i^j(t_1, ..., t_j)) = \bigcup_{k=1}^{j} FV(t_k)$.

(ii) Si φ, ψ son fórmulas, $FV(\neg\varphi) = FV(\varphi)$, $FV(\varphi \square \psi) = FV(\varphi) \cup FV(\psi)$ donde $\square \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$.

(iii) Si x es una variable y φ es una fórmula, $FV(\forall x(\varphi)) = FV(\varphi) \setminus \{x\}$ y $FV(\exists x(\varphi)) = FV(\varphi) \setminus \{x\}$.

Definición 4.1.8. Definimos el conjunto de **variables acotadas** de una fórmula φ, el cual denotaremos por $BV(\varphi)$, como sigue:

(i) Si R_i^j es un símbolo relacional y $t_1, ..., t_j$ son términos, $BV(R_i^j(t_1, ..., t_j)) = \emptyset$.

(ii) Si φ, ψ son fórmulas, $BV(\neg\varphi) = BV(\varphi)$, $BV(\varphi \square \psi) = BV(\varphi) \cup BV(\psi)$ donde $\square \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$.

(iii) Si x es una variable y φ es una fórmula, $BV(\forall x(\varphi)) = BV(\varphi) \cup \{x\}$ y $BV(\exists x(\varphi)) = BV(\varphi) \cup \{x\}$.

Nota. Dado un término t, escribiremos $t(x_0, .., x_k)$ para denotar al término t y para decir que las variables de t son un subconjunto de $\{x_0, ..., x_k\}$. Y dada una fórmula φ, escribiremos $\varphi(x_0, ..., x_k)$ para denotar a la fórmula φ y para decir que las variables libres de φ forman un subconjunto de $\{x_0, ..., x_k\}$.

Definición 4.1.9. Sea $\mathcal{L}$ un lenguaje predicativo.

(i) Diremos que una fórmula φ es **$\mathcal{L}$-libre de cuantificadores** si y sólo si $BV(\varphi) = \emptyset$.

(ii) Dadas φ una $\mathcal{L}$-fórmula libre de cuantificadores y $x_{i_1}, ..., x_{i_k}$ una sucesión finita de variables. Diremos que $\exists x_{i_1} \exists x_{i_2} \cdots \exists x_{i_k}(\varphi)$ es una **$\mathcal{L}$-fórmula existencial** ó **$\exists$-fórmula respecto a $\mathcal{L}$**.

(iii) Dadas φ una $\mathcal{L}$-fórmula libre de cuantificadores y $x_{i_1}, ..., x_{i_k}$ una sucesión finita de variables. Diremos que $\forall x_{i_1} \forall x_{i_2} \cdots \forall x_{i_k}(\varphi)$ es una **$\mathcal{L}$-fórmula universal** ó **$\forall$-fórmula respecto a $\mathcal{L}$**.

(iv) Diremos que una fórmula φ es una **$\mathcal{L}$-sentencia** si y sólo si $FV(\varphi) = \emptyset$.

Definición 4.1.10. (Estructura) Dado un lenguaje predicativo $\mathcal{L} = (\mathcal{S}, \Sigma)$. Una **$\mathcal{L}$-estructura** $\mathcal{A}$ es una tupla $(A, \Sigma, \mathcal{I})$ tal que A es un conjunto no vacío e $\mathcal{I}$ es una función que toma símbolos de $\mathcal{L}$ tal que:

(i) si c es una constante de Σ, $\mathcal{I}(c) \in A$;

(ii) si R_i^j es un símbolo relacional de aridad j de Σ, $\mathcal{I}(R_i^j) \subseteq A^j$;

(iii) si f_i^j es un símbolo funcional de aridad j $\mathcal{I}(f_i^j) : A^j \to A$ es una función.

En lugar de escribir $\mathcal{I}(c), \mathcal{I}(R_i^j), \mathcal{I}(f_i^j)$ para abreviar escribiremos $c^{\mathcal{A}}, (R_i^j)^{\mathcal{A}}, (f_i^j)^{\mathcal{A}}$.
Al conjunto A le llamaremos **dominio** o **universo** de $\mathcal{A}$. A Σ la llamaremos **signatura** de $\mathcal{A}$. Y a $\mathcal{I}$ le llamaremos **función de interpretación** de $\mathcal{A}$.
Definimos la cardinalidad de $\mathcal{A}$, denotada por $|\mathcal{A}|$, como la cardinalidad del conjunto dominio A.
En algunos casos cuando el lenguaje predicativo resulte evidente, nos referiremos a una $\mathcal{L}$-estructura únicamente como estructura.

Notación. A partir de ahora, para una $\mathcal{L}$-estructura $\mathcal{A} = (A, \Sigma, \mathcal{I})$ omitiremos escribir a $\mathcal{I}$ cuando sea evidente o sea especificada, escribiendo únicamente $\mathcal{A} = (A, \Sigma)$. En algunos casos desarrollaremos a Σ y escribiremos $\mathcal{A} = (A, \{c_n : n \in I_C\}, \{R_i^{a_R(i)} : i \in I_R\}, \{f_i^{a_F(i)} : i \in I_F\})$. También abusaremos de la notación y cuando la cardinalidad de alguno de los conjuntos de Σ sea finita y accesible para ser escrita omitiremos los corchetes de conjunto y escribiremos cada símbolo, como en el siguiente ejemplo: $\mathcal{A} = (A, c_0, ..., c_8, R_0^{a_R(0)}, \{f_i^{a_F(i)} : i \in \omega\})$.

Notación. Sea $\mathcal{A} = (A, \mathcal{C}, \mathcal{R}, \mathcal{F})$ una $\mathcal{L}$-estructura.

(i) Dada una tupla finita no vacía $\overline{a} = (a_0, ..., a_{|\overline{a}|-1}) \in A^{<\mathbb{N}}$, denotamos por $(\mathcal{A}, \overline{a})$ a la $\mathcal{L}'$-estructura (A, Σ'), donde $\mathcal{L}' = (\mathcal{S}, \Sigma')$, con $\Sigma' = (C' \cup \{c_0, ..., c_{|\overline{a}|-1}\}, R, F)$, $C' = \{c_{n+|\overline{a}|} : c_n \in C\}$ y $c_i^{(\mathcal{A},\overline{a})} = a_i$ con $0 \leq i \leq |\overline{a}| - 1$ y $c_{|\overline{a}|+n}^{(\mathcal{A},\overline{a})} = c_n^{\mathcal{A}}$ para $c_n \in \mathcal{C}$.

(ii) Sea $Q \subseteq \mathbb{N} \times A^{<\mathbb{N}}$. Denotamos por $(\mathcal{A}, Q)$ a la $\mathcal{L}'$-estructura con dominio A, cuyos símbolos funcionales y constantes son los mismos de $\mathcal{A}$ y se interpretan de la misma forma, sus símbolos relacionales son los símbolos de $\mathcal{A}$ junto con símbolos relacionales $Q_i^{a_Q(i)}$ tales que $Q_i^{a_Q(i)}$ tal que

$$(Q_i^{a_Q(i)})^{(\mathcal{A},Q)} = \{\overline{b} \in A^{a_R(i)} \ : \ (i, \overline{q}) \in Q\}$$

Añadimos a los símbolos relacionales $Q_i^{a_Q(i)}$ de tal forma que los índices de estos y los índices de los símbolos relacionales existentes no se sobrepongan entre sí, también de forma tal que podamos acceder a los índices de los nuevos símbolos relacionales de forma sencilla y el conjunto I_Q de índices de todos los símbolos relacionales sea en efecto un segmento inicial de ω, posiblemente ω.

Definición 4.1.11. Dada una estructura $\mathcal{A} = (A, \Sigma)$ definimos la **estructura relacional inducida por** $\mathcal{A}$, denotada por $\widetilde{\mathcal{A}}$, como la estructura con dominio A y signatura Σ' sin símbolos constantes ni funcionales y tal que si c es una constante de Σ entonces la relación R_k tal que $(R_k)^{\mathcal{A}} = \{c^A\}$ es una relación de Σ' para algún $k \in \omega$ adecuado; si R_j es una relación de Σ entonces también es una relación de Σ' y si f es un funcional de Σ entonces la relación R_k tal que $(R_k)^{\mathcal{A}} = \{(x_1, ..., x_n, y) : f^{\mathcal{A}}(x_1, ..., x_n) = y\}$ es una relación de Σ' para algún $k \in \omega$ adecuado.

Nos interesa decir cuándo una fórmula es «verdadera» en una determinada estructura. Para ello la idea es que dada una fórmula podamos sustituir elementos del dominio de una estructura en las variables libres de la fórmula. Por ello hacemos la siguiente definición.

Definición 4.1.12. Dada una $\mathcal{L}$-estructura $\mathcal{A}$ definimos la **valuación de un término** $t(x_0, ..., x_k)$ respecto a la sucesión finita de elementos de A: $a_0, ..., a_k$, la cual denotamos por $t[a_0, ..., a_k]$, como sigue:

(i) si t es la variable x_i con $0 \leq i \leq k$, $t[a_0, ..., a_k] := a_i$;

(ii) si t es una constante c, $t[a_0, ..., a_k] := c^{\mathcal{A}}$;

(iii) si t es de la forma $f_i^j(t_0, ..., t_j)$, entonces

$$t[a_0, ..., a_k] := (f_i^j)^{\mathcal{A}}(t_1[a_0, ..., a_k], ..., t_j[a_0, ..., a_k])$$

Definición 4.1.13. Dada una $\mathcal{L}$-estructura $\mathcal{A}$, dada una fórmula φ con variables libres, diremos que la sucesión $a_0, ..., a_k$ **satisface la fórmula** $\varphi(x_0, ..., x_k)$ en $\mathcal{A}$, lo cual denotamos por $\mathcal{A} \vDash \varphi[a_0, ..., a_k]$, si y sólo si ocurre alguno de los siguientes casos:

(i) si $\varphi(x_0, ..., x_k)$ es la relación $t_1 = t_2$ con $t_1(x_0, ..., x_k), t_2(x_0, ..., x_k)$ términos. Entonces $\mathcal{A} \vDash \varphi[a_0, ..., a_k]$ si y sólo si $t_1[a_0, ..., a_k] = t_2[a_0, ..., a_k]$;

(ii) si $\varphi(x_0, ..., x_k)$ es la relación $R_i^j(t_1, ..., t_j)$, con $t_i(x_0, ..., x_k)$. Entonces $\mathcal{A} \vDash \varphi[a_0, ..., a_k]$ si y sólo si $(t_1[a_0, ..., a_k], ..., t_j[a_0, ..., a_k]) \in (R_i^j)^{\mathcal{A}}$;

(iii) si $\varphi(x_0, .., x_k)$ es de la forma $\neg\psi$ para alguna fórmula ψ entonces $\mathcal{A} \vDash \varphi[a_0, ..., a_k]$ si y sólo si no ocurre que $\mathcal{A} \vDash \psi[a_0, ..., a_k]$;

(iv) si $\varphi(x_0, .., x_k)$ es de la forma $\psi_1 \wedge \psi_2$ para algunas fórmulas ψ_1, ψ_2 entonces $\mathcal{A} \vDash \varphi[a_0, ..., a_k]$ si y sólo si $\mathcal{A} \vDash \psi_1[a_0, ..., a_k]$ y $\mathcal{A} \vDash \psi_2[a_0, ..., a_k]$;

(v) si $\varphi(x_0, .., x_k)$ es de la forma $\psi_1 \vee \psi_2$ para algunas fórmulas ψ_1, ψ_2 entonces $\mathcal{A} \vDash \varphi[a_0, ..., a_k]$ si y sólo si $\mathcal{A} \vDash \psi_1[a_0, ..., a_k]$ ó $\mathcal{A} \vDash \psi_2[a_0, ..., a_k]$;

(vi) si $\varphi(x_0, .., x_k)$ es de la forma $\psi_1 \Rightarrow \psi_2$ para algunas fórmulas ψ_1, ψ_2 entonces $\mathcal{A} \vDash \varphi[a_0, ..., a_k]$ si y sólo si $\mathcal{A} \vDash (\neg\psi_1)[a_0, ..., a_k]$ ó $\mathcal{A} \vDash \psi_2[a_0, ..., a_k]$;

(vii) si $\varphi(x_0,..,x_k)$ es de la forma $\psi_1 \Rightarrow \psi_2$ para algunas fórmulas ψ_1, ψ_2 entonces $\mathcal{A} \vDash \varphi[a_0,...,a_k]$ si y sólo si $\mathcal{A} \vDash (\psi_1 \Rightarrow \psi_2)[a_0,...,a_k]$ y $\mathcal{A} \vDash (\psi_2 \Rightarrow \psi_1)[a_0,...,a_k]$;

(viii) si $\varphi(x_0,..,x_k)$ es de la forma $\forall x_i(\psi)$ para alguna variable x_i con $0 \leq i \leq k$ y fórmula ψ entonces $\mathcal{A} \vDash \varphi[a_0,...,a_k]$ si y sólo si para toda $a \in A$, $\mathcal{A} \vDash \varphi[a_0,...,a,...,a_k]$. Si $k < i$ entonces $\mathcal{A} \vDash \varphi[a_0,...,a_k]$ si y sólo si $\mathcal{A} \vDash \psi[a_0,...,a_k]$;

(ix) si $\varphi(x_0,..,x_k)$ es de la forma $\exists x_i(\psi)$ para alguna variable x_i con $0 \leq i \leq k$ y fórmula ψ entonces $\mathcal{A} \vDash \varphi[a_0,...,a_k]$ si y sólo si existe $a \in A$, $\mathcal{A} \vDash \varphi[a_0,...,a,...,a_k]$. Si $k < i$ entonces $\mathcal{A} \vDash \varphi[a_0,...,a_k]$ si y sólo si $\mathcal{A} \vDash \psi[a_0,...,a_k]$;

Ahora veremos que la satisfacción de una fórmula $\varphi(x_0,...,x_k)$ sólo depende de a lo más las variables $x_0,...,x_k$, es decir depende de las variables libres de la fórmula.

Proposición 4.1.1. *(a) Sean $t(x_0,...,x_k)$ un término, $n,m \in \omega$ tal que $k \leq n,m$ $a_0,...,a_n$ y $b_0,...,b_m$ dos sucesiones finitas tal que si x_i es variable de t entonces $a_i = b_i$. Entonces $t[a_0,...,a_n] = t[b_0,...,b_m]$.*

(b) Sean $\varphi(x_0,...,x_k)$ una fórmula, $n,m \in \omega$ tal que $k \leq n,m$ $a_0,...,a_n$ y $b_0,...,b_m$ dos sucesiones finitas tal que si x_i es variable libre de φ entonces $a_i = b_i$. Entonces $\mathcal{A} \vDash \varphi[a_0,...,a_n]$ si y sólo si $\mathcal{A} \vDash \varphi[b_0,...,b_m]$.

Demostración. *(a)* Demostremos el resultado por complejidad del término t.

(i) Si t es una variable x_i con $0 \leq i \leq k$ entonces $t[a_0,...,a_n] = a_i = b_i = t[b_0,...,b_m]$.

(ii) Si t es una constante c entonces $t[a_0,...,a_n] = c^{\mathcal{A}} = t[b_0,...,b_m]$.

(iii) Si t es un término de la forma $f_i^j(t_1,...,t_j)$ tales que $t_1,...,t_j$ son términos que satisfacen las condiciones del teorema. Entonces

$$\begin{aligned} t[a_0,...,a_n] &= (f_i^j)^{\mathcal{A}}(t_1[a_0,...,a_n],...,t_j[a_0,...,a_n]) \\ &= (f_i^j)^{\mathcal{A}}(t_1[b_0,...,b_m],...,t_j[b_0,...,b_m]) \\ &= t[b_0,...,b_m] \end{aligned}$$

(b) Demostremos el resultado por complejidad de la fórmula φ.

(i) Si $\varphi(x_0,...,x_k)$ es la relación $t_1 = t_2$ entonces

$$\begin{aligned} \mathcal{A} \vDash \varphi[a_0,...,a_n] &\leftrightarrow t_1[a_0,...,a_n] = t_2[a_0,...,a_n] \\ &\leftrightarrow t_1[a_0,...,a_n] = t_2[b_0,...,b_m] \\ &\leftrightarrow t_1[b_0,...,b_m] = t_2[b_0,...,b_m] \\ &\leftrightarrow \mathcal{A} \vDash \varphi[b_0,...,b_m] \end{aligned}$$

(ii) Si $\varphi(x_0, ..., x_k)$ es de la forma $R_i^j(t_1, ..., t_j)$ entonces

$$\begin{array}{rcl} \mathcal{A} \vDash \varphi[a_0, ..., a_n] & \leftrightarrow & (t_1[a_0, ..., a_n], ..., t_j[a_0, ..., a_n]) \in (R_i^j)^{\mathcal{A}} \\ & \leftrightarrow & (t_1[b_0, ..., b_m], ..., t_j[b_0, ..., b_m]) \in (R_i^j)^{\mathcal{A}} \\ & \leftrightarrow & \mathcal{A} \vDash \varphi[b_0, ..., b_m] \end{array}$$

(iii) Si $\varphi(x_0, ..., x_k)$ es de la forma $\neg\psi$ donde ψ satisface el resultado entonces

$$\begin{array}{rcl} \mathcal{A} \vDash \varphi[a_0, ..., a_n] & \leftrightarrow & \text{no ocurre que } \mathcal{A} \vDash \psi[a_0, ..., a_n] \\ & \leftrightarrow & \text{no ocurre que } \mathcal{A} \vDash \psi[b_0, ..., b_m] \\ & \leftrightarrow & \mathcal{A} \vDash \varphi[b_0, ..., b_m] \end{array}$$

(iv) Si $\varphi(x_0, ..., x_k)$ es de la forma $\psi_1 \wedge \psi_2$ donde ψ_1 y ψ_2 cumplen el resultado entonces

$$\begin{array}{rcl} \mathcal{A} \vDash \varphi[a_0, ..., a_n] & \leftrightarrow & \mathcal{A} \vDash \psi_1[a_0, ..., a_n] \text{ y } \mathcal{A} \vDash \psi_2[a_0, ..., a_n] \\ & \leftrightarrow & \mathcal{A} \vDash \psi_1[b_0, ..., b_m] \text{ y } \mathcal{A} \vDash \psi_2[b_0, ..., b_m] \\ & \leftrightarrow & \mathcal{A} \vDash \varphi[b_0, ..., b_m] \end{array}$$

(v) Si $\varphi(x_0, ..., x_k)$ es de la forma $\psi_1 \vee \psi_2$ donde ψ_1 y ψ_2 cumplen el resultado entonces

$$\begin{array}{rcl} \mathcal{A} \vDash \varphi[a_0, ..., a_n] & \leftrightarrow & \mathcal{A} \vDash \psi_1[a_0, ..., a_n] \text{ ó } \mathcal{A} \vDash \psi_2[a_0, ..., a_n] \\ & \leftrightarrow & \mathcal{A} \vDash \psi_1[b_0, ..., b_m] \text{ ó } \mathcal{A} \vDash \psi_2[b_0, ..., b_m] \\ & \leftrightarrow & \mathcal{A} \vDash \varphi[b_0, ..., b_m] \end{array}$$

(vi) Si $\varphi(x_0, ..., x_k)$ es de la forma $\psi_1 \Rightarrow \psi_2$ donde ψ_1 y ψ_2 cumplen el resultado, la demostración es similar a incisos anteriores.

(vii) Si $\varphi(x_0, ..., x_k)$ es de la forma $\psi_1 \Leftrightarrow \psi_2$ donde ψ_1 y ψ_2 cumplen el resultado, la demostración es similar a incisos anteriores.

(viii) Si $\varphi(x_0, ..., x_k)$ es de la forma $\forall x_i(\psi)$ donde $i \leq k$ y ψ cumple el resultado entonces

$$\begin{array}{rcl} \mathcal{A} \vDash \varphi[a_0, ..., a_n] & \leftrightarrow & \text{para todo } a \in A, \mathcal{A} \vDash \psi[a_0, ..., a, ..., a_n] \\ & \leftrightarrow & \text{para todo } b \in A, \mathcal{A} \vDash \psi[b_0, ..., b, ..., b_m] \\ & \leftrightarrow & \mathcal{A} \vDash \varphi[b_0, ..., b_m] \end{array}$$

Si $k < i$ entonces

$$\begin{array}{rcl} \mathcal{A} \vDash \varphi[a_0, ..., a_n] & \leftrightarrow & \mathcal{A} \vDash \psi[a_0, ..., a_n] \\ & \leftrightarrow & \mathcal{A} \vDash \psi[b_0, ..., b_m] \\ & \leftrightarrow & \mathcal{A} \vDash \varphi[b_0, ..., b_m] \end{array}$$

(ix) Si $\varphi(x_0, ..., x_k)$ es de la forma $\exists x_i(\psi)$, la demostración es similar.

□

A continuación daremos la definición de isomorfismo, la cual será clave dentro de lo que resta de este trabajo. Esta noción intenta decir cuando dos estructuras son iguales salvo por la interpretación de los símbolos que hay en ellas.

Definición 4.1.14. Sean $\mathcal{M}, \mathcal{N}$ dos $\mathcal{L}$-estructuras y $h : M \to N$ una función biyectiva. Diremos que h es un **isomorfismo** entre $\mathcal{M}$ y $\mathcal{N}$ si y sólo si las siguientes ocurren:

(i) $h(c^{\mathcal{M}}) = c^{\mathcal{N}}$ para todo símbolo constante c de $\mathcal{L}$.

(ii) $h(f_i^{\mathcal{M}}(a_1, ..., a_n)) = f_i^{\mathcal{N}}(h(a_1), ..., h(a_n))$ para todo símbolo funcional f_i de $\mathcal{L}$ y para cualesquiera $a_1, .., a_n \in A$.

(iii) $(a_1, ..., a_n) \in R_i^{\mathcal{M}}$ si y sólo si $(h(a_1), ..., h(a_n)) \in R_i^{\mathcal{N}}$ para todo símbolo relacional R_i de $\mathcal{L}$ y para cualesquiera $a_1, .., a_n \in A$.

En este caso diremos que $\mathcal{M}$ y $\mathcal{N}$ son **isomorfas**, lo cual denotamos por $\mathcal{M} \cong \mathcal{N}$.

Una observación inmediata de la definición anterior es la siguiente:

Observación. *(i)* Si $\mathcal{A}$ y $\mathcal{B}$ son estructuras y $f : A \to B$ es un isomorfismo entonces f^{-1} es un isomorfismo de $\mathcal{B}$ a $\mathcal{A}$.

(ii) Si $\mathcal{A}$, $\mathcal{B}$ y $\mathcal{C}$ son estructuras y $f : A \to B$ y $g : B \to C$ son isomorfismos de $\mathcal{A}$ a $\mathcal{B}$ y $\mathcal{B}$ a $\mathcal{C}$ respectivamente. Entonces $f \circ g : A \to C$ es un isomorfismo de $\mathcal{A}$ a $\mathcal{C}$.

Una consecuencia de la observación previa es la siguiente.

Observación. La relación $\cong$ entre estructuras es una relación de equivalencia.

4.1.1. Aritmetización del lenguaje

Dado que una fórmula es una sucesión finita de símbolos, le podemos asignar un número natural tal como se hizo con los programas RAM. Para ello, asociaremos números naturales a los símbolos de un lenguaje predicativo bajo el siguiente estándar:

$$\begin{array}{ccccc} (\to 3 &) \to 5 & , \to 7 & \neg \to 9 & \wedge \to 11 \\ \vee \to 13 & \Rightarrow \to 15 & \Leftrightarrow \to 17 & \forall \to 19 & \exists \to 21 \\ x_0 \to 23 & c_0 \to 25 & f_0 \to 27 & R_0 \to 29 & \\ x_1 \to 31 & c_1 \to 33 & f_1 \to 35 & ... & \end{array}$$

Como mencionamos, cada fórmula bien formada finita es en realidad una sucesión finita de símbolos $x_1...x_n$ respecto a un lenguaje predicativo, por consiguiente podemos ver a esta sucesión como una tupla de números naturales $(x_1, ..., x_n)$ con la codificación anterior. A esta tupla la podemos identificar con un número natural mediante una función biyectiva y computable.

Por otro lado, dada una tupla finita no vacía $(x_1, ..., x_n) \in \omega^{<\mathbb{N}}$, nos interesa saber si dicha tupla corresponde a la codificación de una fórmula finita bien formada. Dado un lenguaje predicativo $\mathcal{L}$, definimos los siguientes predicados:

Predicado	**Significado**
$\mathcal{L} - Term(x)$	$\psi^{-1}(x)$ es la codificación de un $\mathcal{L}$-término
$\mathcal{L} - FormAt(x)$	$\psi^{-1}(x)$ es la codificación de una $\mathcal{L}$-fórmula atómica
$\mathcal{L} - FormLit(x)$	$\psi^{-1}(x)$ es la codificación de una $\mathcal{L}$-literal
$\mathcal{L} - FormSinCuant(x)$	$\psi^{-1}(x)$ es la codificación de una $\mathcal{L}$-fórmula sin cuantificadores
$\mathcal{L} - Exist(x)$	$\psi^{-1}(x)$ es la codificación de una $\mathcal{L}$-fórmula existencial
$\mathcal{L} - Univ(x)$	$\psi^{-1}(x)$ es la codificación de una $\mathcal{L}$-fórmula universal
$\mathcal{L} - Sent(x)$	$\psi^{-1}(x)$ es la codificación de una $\mathcal{L}$-sentencia

Nuestro propósito es mostrar que todos los predicados anteriores son computables. Para ello consideramos programas que toman una tupla finita no vacía de longitud arbitraria e identifica que ésta tenga la sintaxis de un término o de una fórmula de determinada forma según corresponda. La idea de cada uno de dichos programas es que cuentan los paréntesis de las fórmulas que reciben e identifica si lo que existe entre estos paréntesis está bien formado.

A continuación compartimos los programas de cada uno de los programas escritos en Python, el cual es un lenguaje de mayor expresividad que el programas RAM. [1] Hacemos esta breve excepción ya que los algoritmos son suficientemente complejos para que sean escritos con instrucciones RAM. Por ello asumimos que todo lo que es computable con Python es computable con una máquina RAM. [2]

[1] Los programas escritos en Python se encuentran la siguiente liga: `https://github.com/ffeernando/codigo_aritmetizacion_lenguaje`

[2] Lo cual asumimos verdadero ya que con una máquina RAM es posible simular el comportamiento de una computadora cotidiana e incluso todos los programas y lenguajes que se ejecutan en ésta.

4.2. Computabilidad de una estructura

En esta sección diremos cuándo una estructura es computable, ó bien computable respecto a un oráculo $X \subseteq \omega$.

Definición 4.2.1. Sea $\mathcal{A} = (\mathcal{A}, \Sigma)$ una $\mathcal{L}$-estructura. Una **ω-presentación**, o simplemente **presentación**[3], de $\mathcal{A}$ es una estructura $\mathcal{M}$ isomorfa a $\mathcal{A}$ y con dominio ω.

Definición 4.2.2. Dado $B \subseteq \omega$ y una familia de conjuntos $\mathcal{F} = \{A_n : n \in B\}$. Definimos la **suma directa** de $\mathcal{F}$ como sigue

$$\bigoplus \mathcal{F} := \bigoplus_{n \in B} A_n := \{(n, j) : n \in B, j \in A_n\}$$

Si $\mathcal{F}$ es finita, a la suma directa de $\mathcal{F}$ también la denotamos como $A_{k_0} \oplus \cdots \oplus A_{k_n}$ si $B = \{A_{k_0}, ..., A_{k_n}\}$ donde cada $k_i \in B$ para cada $0 \leq i \leq n$.

La intención de la definición anterior es definir una «unión» entre conjuntos de tal forma que podamos diferenciar de entre los elementos de uniendos diferentes.

Definición 4.2.3. *(i)* Dada una ω-presentación $\mathcal{M}$ de una $\mathcal{L}$-estructura $\mathcal{A}$, definimos al conjunto

$$\tau^{\mathcal{M}} := \bigoplus_{i \in I_R} (R_i^{a_R(i)})^{\mathcal{M}} \oplus \bigoplus_{i \in I_F} (f_i^{a_F(i)})^{\mathcal{M}} \oplus \bigoplus_{i \in I_C} \{c_i^{\mathcal{M}}\}$$

Diremos que $\tau^{\mathcal{M}}$ es **la parte estructural de $\mathcal{M}$**.

(ii) Dado $X \subseteq \omega$, diremos que $\mathcal{M}$ es **X-computable** si y sólo si $\boldsymbol{\psi}[\tau^{\mathcal{M}}]$ es X-computable.

A partir de ahora supondremos que nuestras estructuras tienen siempre como primer símbolo relacional a R_0 cuya aridad es 2 y se interpreta como la igualdad usual de números naturales. Dicho esto, otra forma de decir cuándo una estructura es computable es vía su diagrama atómico.

Definición 4.2.4. Sea $\mathcal{M}$ una ω-presentación de una $\mathcal{L}$-estructura $\mathcal{A}$ y $\{\varphi_i^{at} : i \in \omega\}$ una enumeración computable de las $\mathcal{L}$-fórmulas atómicas. Definimos el **diagrama atómico** de $\mathcal{M}$, denotado por $D(\mathcal{M})$ como la sucesión binaria:

$$D(\mathcal{M})(i) = \begin{cases} 1 & \text{si } \mathcal{M} \vDash \varphi_i^{at}[x_j \to j : j \in \omega] \\ 0 & \text{en otro caso} \end{cases}$$

[3]No confundir con la noción de presentación que existe en teoría de grupos.

En la definición anterior, nos referimos con $\varphi_i^{at}(x_{j_1}, ..., x_{j_n})$ para denotar a la fórmula φ_i^{at} y para decir que $x_{j_1}, ..., x_{j_n}$ son todas sus variables y nos referimos con $\mathcal{M} \vDash \varphi_i^{at}[x_j \to j : j \in \omega]$ para decir que $j_1, ..., j_n \in \omega$ satisface $\varphi_i^{at}(x_{j_1}, ..., x_{j_n})$ en $\mathcal{M}$ (lo cual representaríamos por $\mathcal{M} \vDash \varphi_i^{at}[j_1, ..., j_n]$).

Un primer resultado es que en la definición anterior no importa cual enumeración de fórmulas atómicas se tome.

Proposición 4.2.1. *Sean $X \subseteq \omega$ y $\mathcal{M}$ una ω-presentación de una $\mathcal{L}$-estructura $\mathcal{A}$ y E_1, E_2 dos enumeraciones computables de las $\mathcal{L}$-fórmulas, entonces el diagrama atómico de $\mathcal{M}$ respecto de E_1 es X-computable si y sólo si el diagrama atómico de $\mathcal{M}$ respecto de E_2 es X-computable.*

Demostración. Supongamos que el diagrama atómico de $\mathcal{M}$ respecto de E_1 es X-computable y veamos que el diagrama respecto de E_2 también lo es. La otra implicación es análoga.

Definimos un programa tal que dada una entrada i, escribe su fórmula correspondiente a la enumeración E_2, digamos φ. Posteriormente calcula la fórmula φ_0^{at} respecto a la enumeración E_1 y verifica si coincide con la fórmula φ. En caso de que esto no ocurra continúa con la fórmula φ_1^{at} respecto a E_1 y la compara con φ. Este proceso se repite hasta encontrar el índice j para el cual φ_j^{at} coincida con la fórmula φ. Dicho índice existe ya que E_1 y E_2 son enumeraciones del mismo conjunto de $\mathcal{L}$-fórmulas atómicas. Finalmente calculamos la j-ésima entrada del diagrama respecto a E_1. Dicho valor es la i-ésima entrada del diagrama atómico respecto a E_2. Por lo tanto si el diagrama atómico respecto a E_1 es X-computable entonces el diagrama atómico respecto a E_2 debe ser X-computable. □

Dicho lo anterior veremos que el conjunto $\boldsymbol{\psi}[\tau^{\mathcal{M}}]$ y la gráfica de la sucesión $D(\mathcal{M})$ son equivalentes. Abusaremos de la notación y omitiremos el $\boldsymbol{\psi}[]$ en el conjunto $\boldsymbol{\psi}[\tau^{\mathcal{M}}]$ y omitiremos el escribir la gráfica de $D(\mathcal{M})$ para escribir únicamente $D(\mathcal{M})$. Este cambio no afecta los resultados que vamos a mostrar ya que $\boldsymbol{\psi}$ es una función calculable, lo que implica que sus valores se pueden obtener de forma computable, y ya que al tener como oráculo a la gráfica de $D(\mathcal{M})$ tenemos acceso a los valores que toma la función $D(\mathcal{M})$.

Teorema 4.2.1. *Sea $\mathcal{M}$ una ω-presentación de una $\mathcal{L}$-estructura $\mathcal{A}$, entonces $\tau^{\mathcal{M}} \equiv_T D(\mathcal{M})$.*

Demostración. Veamos que $\tau^{\mathcal{M}} \leq_T D(\mathcal{M})$, es decir que $\tau^{\mathcal{M}}$ es $D(\mathcal{M})$-computable.

Definamos un programa **P** tal que dado $z \in \omega$, decodificamos z calculando $\boldsymbol{\psi}^{-1}(z)$. Observamos cuál es el primer elemento de la tupla obtenida. Si es un número mayor o igual a 3 entonces la tupla obtenida no pertenece a $\tau^{\mathcal{M}}$ y nuestro programa devuelve 0.

En otro caso si es igual a 2 la tupla obtenida podría tener codificado a un símbolo constante interpretado. Por ello verificamos si la longitud de la tupla es exactamente 3. En caso de que no sea así inmediatamente devolvemos 0. En caso de que sí identificamos el segundo elemento de la tupla, digamos i y el tercer elemento digamos j. Para que la tupla obtenida se encuentre en $\tau^{\mathcal{M}}$ debería ser cierto que $c_i^{\mathcal{M}} = j$. Por ello consideramos a la fórmula $c_i = x_j$, buscamos su índice y preguntamos a $D(\mathcal{M})$ si ésta fórmula es verdadera en $\mathcal{M}$ cuando x_j es sustituida por j. Devolvemos la respuesta a esta pregunta y finalizamos el programa.

En caso de que la primera coordenada sea igual a 1 la tupla podría tener codificado a una tupla en la gráfica de un funcional interpretado. Para asegurarnos de ello primero obtenemos el valor de la segunda coordenada, digamos i y obtenemos también la cantidad de elementos que existen después de i. Verificamos que existan al menos dos de ellos, en caso de que sólo haya uno, devolvemos 0 y el programa finaliza. En caso de que haya más que uno, obtenemos con la función aridad a_F el valor de $a_F(i)$ y verificamos que la cantidad de elementos que obtuvimos coincida con $a_F(i) + 1$. En caso contrario no vale la pena seguir con el proceso, por lo que devolvemos 0 y finalizamos el programa. En caso de que sí entonces tomamos todos los elementos digamos $j_1, ..., j_n, j_{n+1}$ y consideramos la fórmula atómica $f_i(x_{j_1}, ..., x_{j_n}) = x_{j_{n+1}}$. Obtenemos el índice de esta fórmula y con $D(\mathcal{M})$ nos preguntamos si dicha fórmula es verdadera en $\mathcal{M}$ cuando cada x_{j_k} es sustituido por j_k. Devolvemos la respuesta de esta pregunta y finalizamos el programa.

En caso de que la primera coordenada sea igual a 0 la tupla podría tener codificada a una tupla en un relacional interpretado. Por lo que para verificar esto procedemos de manera similar a la verificación anterior para símbolos funcionales.

Por lo tanto la función inducida por este programa respecto a $D(\mathcal{M})$ como oráculo es la función característica de $\boldsymbol{\psi}[\tau^{\mathcal{M}}]$. Por lo tanto $\tau^{\mathcal{M}}$ es $D(\mathcal{M})$-computable.

Ahora veamos que $D(\mathcal{M}) \leq_T \tau^{\mathcal{M}}$, es decir que $D(\mathcal{M})$ es $\tau^{\mathcal{M}}$-computable.

Definamos un programa tal que dado $i \in \omega$, obtenemos la fórmula atómica φ_i^{at} dada por alguna enumeración computable del conjunto de fórmulas atómicas.

Para cada símbolo constante que ocurre en dicha fórmula atómica identificamos su índice, digamos k. Posteriormente buscamos la primer tupla de tipo $(2, k, j)$ que está en $\tau^{\mathcal{M}}$, usando a $\tau^{\mathcal{M}}$ como oráculo. Observemos que debe haber una ya que todo símbolo constante se interpreta en $\mathcal{M}$. El valor j es el valor de la constante c_i interpretada en $\mathcal{M}$.

Después de esto para cada símbolo funcional que ocurre en φ_i^{at}, obtenemos el índice de dicho símbolo digamos k y consideramos la tupla que está siendo

valuada bajo el símbolo funcional. A partir de esta formemos una nueva tupla en la que sustituimos cada variable por su índice y cada constante por su valor en $\mathcal{M}$ que fue obtenido en el paso anterior. Tendremos una tupla de tipo $(j_1, ..., j_n)$. Posteriormente busquemos la primer tupla de la forma $(1, k, j_1, ..., j_n, j_{n+1})$ que está en $\tau^{\mathcal{M}}$, usando a $\tau^{\mathcal{M}}$ como oráculo. Notemos que debe haber una ya que f_k se interpreta como una función y debe devolver un valor al valuarse en $(j_1, ..., j_n)$. El valor j_{n+1} es el valor del funcional interpretado sobre una tupla interpretada en $\mathcal{M}$ y tal que toda variable fue sustituida por su índice.

Finalmente para el único símbolo relacional posible, obtenemos su índice digamos k. Realizamos las sustituciones de cada constante y cada funcional de modo que al final sólo tengamos los valores de éstos al interpretarse en $\mathcal{M}$ bajo la sustitución de cada variable por su índice. Esto nos formará una tupla finita de números naturales de la forma $(l_1, ..., l_m)$. Finalmente preguntamos, utilizando a $\tau^{\mathcal{M}}$ como oráculo, si la tupla $(0, k, l_1, ..., l_m)$ pertenece a $\tau^{\mathcal{M}}$, devolvemos la respuesta a esta pregunta y finalizamos el programa. Notemos que dicha respuesta determina si φ_i^{at} es verdadera en $\mathcal{M}$ cuando cada una de sus variables se sustituye por su índice.

Por consiguiente la función inducida por este programa respecto a $\tau^{\mathcal{M}}$ como oráculo es la función $D(\mathcal{M})$. Por lo tanto $D(\mathcal{M})$ es $\tau^{\mathcal{M}}$-computable.

Con lo cual concluimos que $\tau^{\mathcal{M}} \equiv_T D(\mathcal{M})$. □

Corolario 4.2.1. *Sean $X \subseteq \omega$ y $\mathcal{M}$ una ω-presentación de una $\mathcal{L}$-estructura $\mathcal{A}$, entonces $\mathcal{M}$ es X-computable si y sólo si $D(\mathcal{M})$ es una función X-computable.*

Demostración. $\mathcal{M}$ es X-computable si y sólo si $\tau^{\mathcal{M}}$ es X-computable si y sólo si $D(\mathcal{M})$ es X-computable. □

En ocasiones nos interesará trabajar con presentaciones de estructura cuyo dominio no sea todo ω. Por ello damos una definición para estructuras con un dominio más pequeño.

Definición 4.2.5. Dada $\mathcal{A}$ una $\mathcal{L}$-estructura, diremos que una $(\subseteq \omega)$-**presentación** de $\mathcal{A}$ es una estructura isomorfa a $\mathcal{A}$ cuyo dominio es un subconjunto de ω.

Observemos que una ω-presentación es una caso particular de una $(\subseteq \omega)$-presentación.

Tal como lo hicimos para presentaciones, también podemos definir el diagrama atómico de una $(\subseteq \omega)$-presentación.

Definición 4.2.6. Sea $\mathcal{M} = (M, \Sigma)$ una $(\subseteq \omega)$-presentación de una $\mathcal{L}$-estructura $\mathcal{A}$ y $\{\varphi_i^{at} : i \in \omega\}$ una enumeración computable de las $\mathcal{L}$-fórmulas atómicas. Definimos el **diagrama atómico** de $\mathcal{M}$, denotado por $D(\mathcal{M})$ como la sucesión binaria tal que:

$$D(\mathcal{M})(i) = \begin{cases} 1 \quad \text{si} & \mathcal{M} \vDash \varphi_i^{at}[x_j \to j : j \in \omega] \text{ y} \\ & \forall j \in \omega \ (x_j \in FV(\varphi_i^{at}) \Rightarrow j \in M) \\ 0 & \text{en otro caso} \end{cases}$$

Observemos que la definición es análoga al de una ω-presentación. También notemos que para una ω-presentación la nueva condición $\forall j \in \omega \ (x_j \in FV(\varphi_i^{at}) \Rightarrow j \in M)$ siempre se cumple ya que $M = \omega$.

También de manera similar al caso de una ω-presentación $\mathcal{M}$, definimos la parte estructural de $\mathcal{M}$. En este caso el conjunto obtenido seguirá siendo un subconjunto de tuplas finitas de $\omega^{<\mathbb{N}}$ ya que el conjunto dominio de $\mathcal{M}$ es un subconjunto de ω.

A continuación enunciamos la versión para ($\subseteq \omega$)-presentaciones del teorema 4,2,1.

Teorema 4.2.2. *Sea $\mathcal{M}$ una ($\subseteq \omega$)-presentación de una estructura $\mathcal{A}$. Entonces $D(\mathcal{M}) \equiv_T M \oplus \tau^{\mathcal{M}}$.*

Demostración. Veamos que $D(\mathcal{M}) \leq_T M \oplus \tau^{\mathcal{M}}$, es decir que $D(\mathcal{M})$ es $M \oplus \tau^{\mathcal{M}}$-computable.

Para ello basta con modificar el programa del teorema anterior que verifica que $D(\mathcal{M}) \leq \tau^{\mathcal{M}}$ agregando un paso en el que verifica que todos los índices de las variables de la fórmula en cuestión pertenezcan a M y además que en las búsquedas de los valores de las constantes y funcionales los elementos obtenidos pertenezcan también a M. En este programa utilizamos a $M \oplus \tau^{\mathcal{M}}$ como oráculo. Por lo tanto $D(\mathcal{M})$ es $M \oplus \tau^{\mathcal{M}}$-computable.

Para ver que $\mathcal{M} \oplus \tau^{\mathcal{M}} \leq_T D(\mathcal{M})$, modificamos el programa correspondiente al teorema anterior que verifica que $\tau^{\mathcal{M}} \leq_T D(\mathcal{M})$ para incluir un proceso en el que en caso de preguntarnos por una tupla de la forma $(0, j)$, busquemos el índice de la fórmula $x_j = x_j$ y preguntemos a $D(\mathcal{M})$ si esta fórmula es verdadera cuando se sustituye x_j por j y si $j \in M$. Como siempre ocurre que $j = j$ entonces la respuesta que obtengamos de la pregunta anterior nos dirá si $j \in M$. Por lo que en este caso devolvemos la respuesta obtenida.

En caso de que nos preguntemos por una tupla en $\tau^{\mathcal{M}}$ al programa mencionado anteriormente le agregamos un proceso en el que verifica si los elementos de la tupla pertenecen a M, lo cual podemos conseguir de manera similar a como hemos mencionado con anterioridad y además que en caso de que sea así, también en la búsqueda de constantes y funcionales interpretados nos cercioremos que los elementos sobre los que estamos buscando estén en M, lo cual podemos lograr de

la misma forma antes mencionada.

Dicho esto concluimos que $D(\mathcal{M}) \equiv_T M \oplus \tau^{\mathcal{M}}$. □

Una observación que también es importante es la siguiente:

Observación. Para toda ($\subseteq \omega$)-presentación infinita $\mathcal{M}$ de la forma $(M; \{R_i : i \in I_R\})$, existe una ω-presentación $\mathcal{A}$ de $\mathcal{M}$.

Demostración. Supongamos que $M = \{m_i : i \in \omega\}$ donde para cada $i \in \omega$, $m_i < m_{i+1}$. Sea $\mathcal{A} = (\omega; \{R_i : i \in I_R\})$ tal que

$$R_i^{\mathcal{A}} = \{(n_1, ..., n_{a_R(i)}) \in \omega^{a_R(i)} : (m_{n_1}, ..., m_{n_{a_R(i)}}) \in R_i^{\mathcal{M}}\}$$

Sea $h : A \to M$ tal que $h(i) = m_i$. Observemos que h es una función biyectiva y además por las definiciones de las interpretaciones de los símbolos R_i tenemos que h es un isomorfismo entre $\mathcal{A}$ y $\mathcal{M}$. Por lo tanto $\mathcal{A}$ es una ω-presentación de $\mathcal{M}$. □

4.2.1. Estructura relacional inducida por una estructura y enumeración de una estructura

Veamos que, en efecto, el diagrama atómico de $\mathcal{M}$ es Turing equivalente al diagrama de su estructura relacional inducida $\widetilde{\mathcal{M}}$. Para ello demostremos el siguiente lema:

Lema 4.2.1. *Sean $\mathcal{M}$ una estructura y $\widetilde{\mathcal{M}}$ su estructura relacional. Existe una función computable $f : \omega \to \omega$ tal que para cada fórmula libre de cuantificadores φ_i^{lc} en el lenguaje de $\mathcal{M}$ existe una fórmula existencial $\widetilde{\varphi}_{f(i)}^{\exists}$ en el lenguaje de $\widetilde{\mathcal{M}}$ tal que $\mathcal{M} \vDash \varphi_i[x_j \to j : j \in \omega]$ si y sólo si $\widetilde{\mathcal{M}} \vDash \widetilde{\varphi}_{f(i)}^{\exists}[x_j \to j : j \in \omega]$.*

Demostración. Demostremos el resultado sobre la complejidad de φ. Para ello primero demostremos el resultado para fórmulas atómicas.

Dada una fórmula atómica, daremos un proceso que se puede seguir de forma computable para obtener el índice de la fórmula existencial que se requiere.

Sea φ_i una fórmula atómica de la forma $R_k(t_1, ..., t_s)$ con $t_1, ..., t_s$ términos. Si el término t_j tiene variables, supongamos que $x_{q_1}, ..., x_{q_n}$ son sus variables y si tiene constantes supongamos que $c_{s_1}, ..., c_{s_m}$ son sus constantes.

Cada término t_j debe tener al menos una variable o al menos una constante y puede tener sólo constantes o sólo variables o ambas. También para cada término t_j existe una cantidad finita de apariciones de símbolos funcionales, digamos z, para poder conformar a t_j y digamos que los símbolos son $f_{r_1}, ..., f_{r_k}$. Observemos que el número de apariciones de símbolos no necesariamente es idéntico al número

de símbolos funcionales distintos que aparecen ya que un símbolo funcional puede aparecer más de una vez. Es posible también que t_j no tenga símbolos funcionales.

Observemos que un programa puede identificar a la perfección todos los símbolos de la fórmula, más aún puede obtener sus índices.

Ahora bien, recordemos que a una constante c_i en el lenguaje de $\mathcal{M}$ le corresponde un símbolo relacional R_{3i+1} de aridad uno que contiene únicamente a $c_i^{\mathcal{A}}$ en el lenguaje de $\widetilde{\mathcal{M}}$. De la misma forma a un símbolo funcional f_i le corresponde un símbolo relacional R_{3i+2} que se interpreta en $\mathcal{A}$ como la gráfica de $f_i^{\mathcal{A}}$. Y a un símbolo relacional R_i le asocia un símbolo relacional R_{3i} que se interpreta en $\mathcal{M}$ de la misma forma que R_i. Por consiguiente, dicho programa puede generar los índices correspondientes de los símbolos constantes, funcionales y relacionales que le corresponden a la fórmula original.

Definiremos la fórmula requerida siguiendo el orden que se presenta a continuación:

Si φ tiene constantes, se sustituye la constante c_{s_i} por x_{q_n+i} y agregamos la fórmula $R_{3s_i+1}(x_{q_n+i})$. Al terminar hacemos la conjunción de todas estas fórmulas la cual denotamos por φ^c, y cuantificamos toda la fórmula a construir con $\exists x_{q_n+1}\exists x_{q_n+2}\cdots\exists x_{q_n+m}$. Si no hay constantes vamos al siguiente paso.

Si φ tiene símbolos funcionales, comenzamos por sustituir aquellos símbolos funcionales, digamos f_{r_i}, cuyos términos sean únicamente variables, digamos $f_{r_i}(x_{l_1},...,x_{l_{a_F(r_i)}})$, por una variable x_p comenzando por x_{q_n+m+1} y tal que no se haya utilizado hasta el momento y agregamos la fórmula $R_{3i+2}(x_{l_1},...,x_{l_{a_F(i)}},x_p)$. Al terminar hacemos la conjunción de todas estas fórmulas la cual denotamos por φ^f y cuantificamos toda la fórmula que construiremos con $\exists x_{q_n+m+1}\exists x_{q_n+m+2}\cdots\exists x_{q_n+m+z}$. Si φ no tiene símbolos funcionales vamos al siguiente paso.

Sustituimos el símbolo relacional R_i por R_{3i}.

Finalmente si φ contaba con al menos una constante o un símbolo funcional tendremos una fórmula delimitada con cuantificadores existenciales que consiste de la relación R_{3i} sobre términos que son únicamente variables en conjunción con φ^c y φ^f, dependiendo si ambos existen o sólo uno de ellos según sea el caso. Si φ no tenía constantes ni símbolos funcionales entonces únicamente tendremos a la fórmula que consiste del símbolo relacional R_{3i} sobre variables. A esta fórmula la delimitamos con un cuantificador existencial sobre la variable x_{q_n+1}. Observemos que esta cuantificación no afecta ninguna variable de la fórmula que es cuantificada.

Notemos que la fórmula obtenida es existencial en el lenguaje de $\widetilde{\mathcal{M}}$. Además notemos que las sustituciones que hicimos se pueden replicar de forma computable, lo que implica que dado el índice de φ_i podamos obtener de forma computable el índice de la fórmula obtenida la cual denotamos por $\widetilde{\varphi}^{\exists}_{f(i)}$.

Es evidente que $\mathcal{M} \vDash \varphi_i[x_j \to j : j \in \omega]$ si y sólo si $\widetilde{\mathcal{M}} \vDash \widetilde{\varphi}^{\exists}_{f(i)}[x_j \to j : j \in \omega]$.

Observemos que si $\varphi = \neg\eta_i$ donde η_i la podemos convertir en una conjunción de fórmulas relacionales junto con η_i como en el caso anterior a excepción de que cuantificamos toda la fórmula antes del símbolo $\neg$.

De la misma foma si $\varphi = \psi_1 \square \psi_2$ donde $\square \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$ convertimos las fórmulas ψ_1 y ψ_2 que contengan únicamente variables; hacemos la conjunción de dicha transformación con las fórmulas relacionales que obtengamos por símbolos constantes y/o funcionales y finalmente cuantificamos existencialmente toda la fórmula según corresponda.

Todas las conversiones se pueden realizar de forma computable ya que podemos realizar la conversión comenzando con las fórmulas atómicas siguiendo el proceso que describimos para ese caso.

□

Lema 4.2.2. *Sean $\mathcal{M}$ una estructura, $\widetilde{\mathcal{M}}$ su estructura relacional, $\{\varphi^{\exists}_i : i \in \omega\}$, $\{\widetilde{\varphi}^{\exists}_i : i \in \omega\}$ conjuntos de las fórmulas existenciales de $\mathcal{M}$ y $\widetilde{\mathcal{M}}$ respectivamente. Entonces*

$$\{i \in \omega : \mathcal{M} \vDash \varphi^{\exists}_i[x_j \to j : j \in \omega]\} \equiv_m \{i \in \omega : \widetilde{\mathcal{M}} \vDash \widetilde{\varphi}^{\exists}_i[x_j \to j : j \in \omega]\}$$

Demostración. Veamos que $\{i \in \omega : \mathcal{M} \vDash \varphi^{\exists}_i[x_j \to j : j \in \omega]\} \leq_m \{i \in \omega : \widetilde{\mathcal{M}} \vDash \widetilde{\varphi}^{\exists}_i[x_j \to j : j \in \omega]\}$. Sea $f : \omega \to \omega$ la función del lema anterior.

Ahora bien, dada una fórmula $\varphi^{\exists}_k$ en el lenguaje de $\mathcal{M}$, por definición ésta es de la forma $\exists x_{i_1} \cdots \exists x_{i_k} \psi_j$ donde ψ_j es una fórmula libre de cuantificadores. Por el lema anterior para ψ_j existe una fórmula existencial en el lenguaje de $\widetilde{\mathcal{M}}$, $\widetilde{\psi}^{\exists}_{f(j)}$ tal que $\mathcal{M} \vDash \psi_j[x_j \to j : j \in \omega]$ si y sólo si $\widetilde{\mathcal{M}} \vDash \widetilde{\psi}^{\exists}_{f(j)}[x_j \to j : j \in \omega]$. A la fórmula anterior ψ_j le hacemos también la modificación de que las constantes y funcionales que cambiará por variables sean distintas de las variables $x_{i_1}, ..., x_{i_k}$. Esto nos produce el índice $g(i)$ de la fórmula obtenida. Observemos también que la fórmula obtenida es existencial y además satisface que $\mathcal{M} \vDash \varphi_i[x_j \to j : j \in \omega]$ si y sólo si $\widetilde{\mathcal{M}} \vDash \widetilde{\varphi}^{\exists}_{g(i)}[x_j \to j : j \in \omega]$.

Como g es computable entonces concluimos que

$$\{i \in \omega : \mathcal{M} \vDash \varphi^{\exists}_i[x_j \to j : j \in \omega]\} \leq_m \{i \in \omega : \widetilde{\mathcal{M}} \vDash \widetilde{\varphi}^{\exists}_i[x_j \to j : j \in \omega]\}$$

Ahora veamos que

$$\{i \in \omega : \widetilde{\mathcal{M}} \vDash \widetilde{\varphi}_i^{\exists}[x_j \to j : j \in \omega]\} \leq_m \{i \in \omega : \mathcal{M} \vDash \varphi_i^{\exists}[x_j \to j : j \in \omega]\}$$

Para ello a una fórmula atómica con símbolo relacional de la forma R_{3i} con $i \in \omega$ le asignamos exactamente la misma fórmula pero cambiamos por R_i, el cual es un símbolo en el lenguaje de $\mathcal{M}$. Si tenemos una fórmula atómica de la forma $R_{3i+1}(x_j)$ para algunos $i \in \omega$ y x_j una variable, sustituimos esta fórmula por $c_i = x_j$ la cual es una fórmula en el lenguaje de $\mathcal{M}$. Si tenemos una fórmula atómica $R_{3i+2}(x_{i_1}, ..., x_{i_l}, x_k)$ para algunas variables $x_{i_1}, ..., x_{i_l}, x_k$ entonces la sustituimos por $f_i(x_{i_1}, ..., x_{i_l}) = x_k$.

Finalmente realizamos todas las sustituciones de la forma que se acaba de mencionar de fórmulas atómicas de una fórmula existencial $\widetilde{\varphi}_i^{\exists}$. Tras esto obtenemos una fórmula existencial $\varphi_{g(i)}^{\exists}$ en el lenguaje de $\mathcal{M}$ tal que el índice $g(i)$ se puede obtener de forma computable ya que las sustituciones se pueden realizar únicamente tomando en cuenta los índices de los símbolos relacionales, que nos dicen qué sustitución hacer, y sus componentes. Más aún $\widetilde{\mathcal{M}} \vDash \widetilde{\varphi}_i^{\exists}[x_j \to j : j \in \omega]$ si y sólo si $\mathcal{M} \vDash \varphi_{g(i)}^{\exists}[x_j \to j : j \in \omega]$.

Por lo tanto,

$$\{i \in \omega : \widetilde{\mathcal{M}} \vDash \widetilde{\varphi}_i^{\exists}[x_j \to j : j \in \omega]\} \leq_m \{i \in \omega : \mathcal{M} \vDash \varphi_i^{\exists}[x_j \to j : j \in \omega]\}$$

Así concluimos que

$$\{i \in \omega : \mathcal{M} \vDash \varphi_i^{\exists}[x_j \to j : j \in \omega]\} \equiv_m \{i \in \omega : \widetilde{\mathcal{M}} \vDash \widetilde{\varphi}_i^{\exists}[x_j \to j : j \in \omega]\}$$

□

Teorema 4.2.3. *Sean $\mathcal{M}$ una estructura y $\widetilde{\mathcal{M}}$ su $\mathcal{L}$-estructura relacional inducida. Entonces $D(\mathcal{M}) \equiv_T D(\widetilde{\mathcal{M}})$.*

Demostración. Veamos que $D(\mathcal{M})$ es $D(\widetilde{\mathcal{M}})$-computable. Sea $f : \omega \to \omega$ la función computable construida en el lema anterior que hace que $\{i \in \omega : \mathcal{M} \vDash \varphi_i^{\exists}[x_j \to j : j \in \omega]\} \leq_m \{i \in \omega : \widetilde{\mathcal{M}} \vDash \widetilde{\varphi}_i^{\exists}[x_j \to j : j \in \omega]\}$.

Sea $\varphi_i^{at}(x_0, ..., x_k)$ una fórmula atómica en el lenguaje de $\mathcal{M}$ con exactamente $x_0, ..., x_k$ como variables libres. Cuantificamos dicha fórmula existencialmente para obtener una fórmula de la forma $\exists x_{k+1}\varphi_i^{at}$, la cual es una fórmula existencial en el lenguaje de $\mathcal{M}$. Sea $\widetilde{\varphi}_{f(i)}^{\exists}$ la fórmula existencial en el lenguaje de $\widetilde{\mathcal{M}}$ que es producida por f. Recordemos que f agrega más cuantificadores existenciales dependiendo si φ_i^{at} tiene constantes y/o términos.

Primero identificamos si tenemos símbolos relacionales de la forma $R_{3i+1}(x_j)$. En caso de que sea así, verificamos variable por variable si R_{3i+1} es verdadera tras

sustituir dicha variable por su índice. Esto lo hacemos con $D(\widetilde{\mathcal{M}})$. Guardamos dicha variable. Repetimos este proceso con las demás fórmulas de este tipo que existan. Sustituimos las variables obtenidas por las variables acotadas con las que se producieron éstas y eliminamos los cuantificadores de dichas variables acotadas.

Posteriormente identificamos si tenemos símbolos relacionales de la forma R_{3i+2}. Comenzamos por identificar a aquellos símbolos tal que todas sus variables son libres a excepción de la última. Éstas existen ya que todo término se construye en su mínima expresión de variables y/o constantes (que de existir, en el proceso anterior las sustituimos por las variables adecuadas). Con éstas buscamos variable por variable cuál es la que tras ser sustituida por la última variable genera una fórmula atómica que es verdadera tras sustituir cada variable por su índice, esto lo hacemos con $D(\widetilde{\mathcal{M}})$. Ésta debe existir por la definición del símbolo relacional.

Al momento de encontrar a dicha variable, la sustituimos por todas las ocurrencias, en la fórmula, de la variable acotada existencialmente que la genera. Y eliminamos el cuantificador de dicha variable. Continuamos con el proceso con los demás símbolos relacionales de este tipo. Al final habremos eliminado todos los cuantificadores existenciales excepto al que añadimos artificialmente $\exists x_{k+1}$ y tendremos una fórmula relacional R_{3i} cuyas variables ya han sido elegidas de tal forma que al ser sustituidas por sus índices reflejan la sustituición de un término o constante.

Verificamos con $D(\widetilde{\mathcal{M}})$ si dicha fórmula es verdadera. Si es verdadera, entonces la fórmula existencial $\widetilde{\varphi}^{\exists}_{f(i)}$ es verdadera lo cual implica que $\exists x_{k+1}\varphi$ es verdadera. De esto podemos concluir que φ es verdadera en $\mathcal{M}$ tras sustituir sus variables por sus índices. De igual manera, podemos concluir que φ es falsa si la fórmula obtenida es falsa. Por lo tanto, $D(\mathcal{M})$ es $D(\widetilde{\mathcal{M}})$-computable.

Ahora veamos que $D(\widetilde{\mathcal{M}})$ es $D(\mathcal{M})$-computable. Sea $\widetilde{\varphi_i}^{at}$ una fórmula atómica en el lenguaje de $\widetilde{\mathcal{M}}$. Si $\widetilde{\varphi}_i$ tiene un símbolo relacional R_{3j}, verificamos de forma $D(\mathcal{M})$ computable si la fórmula atómica R_j es verdadera después de sustituir las variables de $\widetilde{\varphi}_i$ por sus índices. Dicha fórmula es verdadera si y sólo si $\widetilde{\varphi}_i$ es verdadera. Si tiene un símbolo relacional $R_{3i+1}(x_k)$ entonces buscamos verificamos con $D(\mathcal{M})$ si la fórmula $c_i = x_k$ es verdadera en $\mathcal{M}$ tras sustituir x_k por k. Ésta es verdadera si y sólo si $R_{3i+1}(x_k)$ lo es. Ahora si tiene un símbolo relacional $R_{3i+2}(x_{i_1}, ..., x_{i_k}, x_{i_{k+1}})$ entonces verificamos si la fórmula $f_i(x_{i_1}, ..., x_{i_k}) = x_{i_{k+1}}$ es verdadera en $\mathcal{M}$, en $D(\mathcal{M})$, tras sustituir las variables por sus índices. Ésta fórmula es verdadera si y sólo si la fórmula $R_{3i+2}(x_{i_1}, ..., x_{i_k}, x_{i_{k+1}})$ lo es. Concluimos que $D(\widetilde{\mathcal{M}})$ es $D(\mathcal{M})$-computable.

Por lo tanto, $D(\mathcal{M}) \equiv_T D(\widetilde{\mathcal{M}})$.

□

Del teorema anterior podemos concluir que si hablamos de una estructura en términos de su grado de Turing, no importa si elegimos ésta o su estructura relacional inducida. Por lo tanto, a partir de ahora daremos resultados considerando estructuras con lenguajes relacionales, es decir a aquellas cuyos lenguajes, si tienen símbolos, únicamente tienen símbolos relacionales. Aunque en algunos casos, al momento de enunciar una estructura, seguiremos enunciando sus constantes y funcionales si tiene.

Proposición 4.2.2. *Dado un lenguaje relacional finito $\mathcal{L}$, el número de $\mathcal{L}$-fórmulas atómicas en las variables $x_0, ..., x_{k-1}$ es finito.*

Demostración. En efecto, para ello supongamos que $R_0, R_1, ..., R_n$ son todos los símbolos relacionales de $\mathcal{L}$ y $a_R(i)$ es la aridad del símbolo relacional R_i. Entonces la cantidad de $\mathcal{L}$-fórmulas atómicas distintas en las variables $x_0, ..., x_{k-1}$ para un mismo símbolo relacional R_i es $k^{a_R(i)}$. Luego, la cantidad total de $\mathcal{L}$-fórmulas atómicas en las variables $x_0, ..., x_{k-1}$ es $\sum_{i=0}^{n} k^{a_R(i)}$, lo cual prueba esta proposición. □

De ahora en adelante supondremos que l_k es el total de $\mathcal{L}$-fórmulas atómicas en las variables $x_0, ..., x_{k-1}$. De la demostración anterior observemos también que la función que a cada $k \in \omega$ le asigna l_k es computable.

Proposición 4.2.3. *Dado un lenguaje relacional finito $\mathcal{L}$, existe una enumeración de las $\mathcal{L}$-fórmulas atómicas $\{\varphi_i^{at} : i \in \omega\}$ tal que las fórmulas que tienen como variables libres a x_0 van primero, posteriormente aquellas que tienen a x_0 y/o x_1 y así sucesivamente.*

Demostración. Recordemos que el conjunto de $\mathcal{L}$-fórmulas atómicas es computable enumerable por lo que dicho conjunto podemos enumerarlo por una función computable, digamos f.

Como $\mathcal{L}$ es finito, existe una cantidad finita de símbolos relacionales. Por lo tanto, existe una cantidad finita de fórmulas atómicas en las variables $x_0, ..., x_{k-1}$ para cada $k \in \omega$, digamos l_k.

Construimos un programa que busca primero las fórmulas atómicas cuya única variable es x_0 y verifica que ninguna fórmula anterior coincide con la fórmula encontrada. Cuando hayamos obtenido l_0 de éstas fórmulas continuamos con las fórmulas que tienenúnicamente las variables x_0 y x_1. Verificamos que no sea ninguna fórmula anterior. Continuamos este proceso con las demás variables. Esto nos da una enumeración de las $\mathcal{L}$-fórmulas atómicas por sus variables. □

De ahora en adelante dado un lenguaje relacional $\mathcal{L}$ al momento de obtener una enumeración de sus $\mathcal{L}$-fórmulas atómicas, éstas se encontrarán ordenadas como se

indica en la proposición anterior.

A continuación definiremos una noción de diagrama atómico relativo a una tupla de elementos de la estructura.

Definición 4.2.7. Dados una $\mathcal{L}$-estructura $\mathcal{M}$ y $\overline{a} = (a_0, ..., a_{s-1}) \in M^s$ definimos **el diagrama atómico de $\overline{a}$ en $\mathcal{M}$**, denotado como $D_{\mathcal{M}}(\overline{a})$ como la sucesión de longitud l_s (donde l_s es el número de fórmulas atómicas en las variables $x_0, ..., x_{s-1}$ y en los primeros s símbolos relacionales de $\mathcal{M}$) tal que:

$$D_{\mathcal{M}}(\overline{a})(i) = \begin{cases} 1 & \text{si } \mathcal{M} \vDash \varphi_i^{at}[x_j \to a_j : j < s] \\ 0 & \text{otro caso} \end{cases}$$

Una observación importante es las siguiente:

Observación. Sean $\mathcal{M}$ una estructura, $\sigma \in 2^{<\mathbb{N}}$ una sucesión finita no vacía de 0's y 1's y $\overline{a} = (a_0, ..., a_{k-1}) \in M^k$ con $k \in \mathbb{N}$ tal que $l_k \geq |\sigma|$. Entonces existe una fórmula libre de cuantificadores $\varphi^{lc}_{t(\sigma)}$ tal que $\mathcal{M} \vDash \varphi^{lc}_{t(\sigma)}[x_j \to a_j : j < k]$ si y sólo si $\sigma \subseteq D_{\mathcal{M}}(\overline{a})$.

Demostración. Abreviamos $\overline{x}$ por $(x_0, ..., x_{k-1})$. La fórmula $\varphi^{lc}_{t(\sigma)}(\overline{x})$ definida como

$$\Big(\bigwedge_{(i<|\sigma|)\wedge(\sigma(i)=1)} \varphi_i^{at}(\overline{x})\Big) \wedge \Big(\bigwedge_{(i<|\sigma|)\wedge(\sigma(i)=0)} \neg\varphi_i^{at}(\overline{x})\Big)$$

satisface lo requerido. □

A continuación presentamos una forma de construir estructuras que nos será muy útil en demostraciones posteriores.

Definición 4.2.8. Dados un lenguaje relacional $\mathcal{L}$, una ω-presentación $\mathcal{M} = (M, \{R_i : i \in I_R\})$ de una estructura $\mathcal{A}$ y una función sobreyectiva $f : \omega \to \omega$, definimos el **pull-back** de $\mathcal{M}$, denotado por $f^{-1}(\mathcal{M})$ como la estructura $(X, \{R_i : i \in I_R\})$, donde:

(i) $X = \{x_i : i \in \omega\}$ tal que $x_i < x_j$ si y sólo si $i < j$ y si $f(x_i) = n$ entonces $x_i = \text{mín}\{m \in \omega : \ f(m) = n\}$.

(ii) $R_0^{\mathcal{M}}$ se interpreta como la igualdad usual de los números naturales.

(iii) Para cada $i \in I_R$, $i \neq 0$,

$$R_i^{f^{-1}(\mathcal{M})} := \{(k_1, ..., k_{a_R(i)}) \in \omega^{a_R(i)} : \ (f(k_1), ..., f(k_{a_R(i)})) \in R_i^{\mathcal{M}}\}$$

Una observación inmediata de la definición anterior es que $f \restriction X$ es un isomorfismo entre $f^{-1}(\mathcal{M})$ y $\mathcal{M}$. Por lo tanto $f^{-1}(\mathcal{M})$ es una $(\subseteq \omega)$-presentación de $\mathcal{M}$. Más aún como f es sobreyectiva tenemos que $f^{-1}(\mathcal{M})$ es infinita, es decir

su dominio es infinito.

Por la observación 4,2 existe una ω-presentación de $\mathcal{N}$ de $f^{-1}(\mathcal{M})$. Si $h : \omega \to X$ es el isomorfismo de dicha observación que va de $\mathcal{N}$ a $f^{-1}(\mathcal{M})$, entonces $g := h \circ f \upharpoonright X$ es un isomorfismo de $\mathcal{N}$ a $\mathcal{M}$. Por lo tanto $\mathcal{N}$ es una ω-presentación de $\mathcal{M}$ que satisface las siguientes propiedades:

(i) $R_0^{\mathcal{N}}$ se interpreta como la igualdad usual de los números naturales

(ii) Para cada $i \in I_R$, $i \neq 0$,

$$R_i^{\mathcal{N}} := \{(k_1, ..., k_{a_R(i)}) \in \omega^{a_R(i)} : \ (g(k_1), ..., g(k_{a_R(i)})) \in R_i^{\mathcal{M}}\}$$

Por lo tanto sin pérdida de generalidad podemos suponer que el pull-back de una estructura $\mathcal{M}$ respecto a una función sobreyectiva f es una ω-presentación de $\mathcal{M}$.

A continuación mostramos la relación que existe entre el grado de Turing de un pull-back de una estructura $\mathcal{M}$ y $\mathcal{M}$.

Proposición 4.2.4. *Dados un lenguaje relacional $\mathcal{L}$, una $\mathcal{L}$-estructura $\mathcal{M}$, $(M, \{R_i^{\mathcal{M}} : i \in I_R\})$ y una función sobreyectiva $f : \omega \to \omega$ tenemos que*

$$D(f^{-1}(\mathcal{M})) \leq_T f \oplus D(\mathcal{M})$$

Demostración. Sea φ_i^{at} atómica en el lenguaje de $f^{-1}(\mathcal{M})$ de la forma $R_j(x_{k_1}, ..., x_{k_{a_R(j)}})$. Con f calculamos los valores $f(x_{k_l})$ para cada $1 \leq l \leq a_R(j)$ para obtener la fórmula $R_j(x_{f(k_1)}, ..., x_{f(x_{a_R(j)})})$. Obtenemos el índice de esta fórmula y verificamos con $D(\mathcal{M})$ si dicha fórmula es verdadera tras sustituir sus variables por sus índices. En caso de que así sea tendríamos que $(f_{k_1}, ..., f_{k_{a_R(j)}}) \in R_i^{\mathcal{M}}$, lo cual implica que $(k_1, ..., k_{a_R(j)}) \in R_i^{f^{-1}(\mathcal{M})}$, que implica que la fórmula φ_i^{at} es verdadera tras sustituir sus variables por sus índices y por tanto obtenemos que $D(f^{-1}(\mathcal{M}))(i) = 1$. Si la respuesta es no, con un argumento similar podemos concluir que $D(f^{-1}(\mathcal{M}))(i) = 0$.

Por lo tanto, $D(f^{-1}(\mathcal{M}))$ es $f \oplus D(\mathcal{M})$-computable, es decir $D(f^{-1}(\mathcal{M})) \leq_T f \oplus D(\mathcal{M})$. □

Capítulo 5

Mismas estructuras, diferentes propiedades de computabilidad

En secciones pasadas hemos asignado un grado de Turing a una estructura $\mathcal{M}$, a saber $D(\mathcal{M})$. Por otro lado hemos definido una noción de similitud entre estructuras como lo es el isomorfismo. Esta noción nos permite decir que dos estructuras $\mathcal{A}$ y $\mathcal{B}$ son iguales salvo la interpretación de los símbolos y sus conjuntos dominio. Sin embargo, el hecho de que dos estructuras sean isomorfas no significa que tengan las mismas propiedades de computabilidad; por ejemplo una puede ser computable y la otra no; o pueden tener el mismo grado de Turing pero tienen diferentes propiedades de computabilidad, lo cual como causa puede ser que entre ellas no pueda haber un isomorfismo computable.

Primero mostraremos una condición que implica que una estructura puede tener estructuras isomorfas de distintos grados de Turing. Posteriormente daremos un ejemplo de dos estructuras isomorfas tales que no puede existir un isomorfismo computable entre ellas y después de esto daremos caracterizaciones para que esto ocurra.

5.1. Grados de Turing de ω-presentaciones de una estructura

Para demostrar que existen estructuras que pueden tener ω-presentaciones de distintos grados de Turing, introduciremos una condición para producir automorfismos en una estructura y veremos que aquellas que no cumplen dicha condición son las que necesitamos.

Definición 5.1.1. Diremos que una $\mathcal{L}$-estructura $\mathcal{A} = (A; \Sigma)$ es **trivial** si y sólo si existen $a_1, ..., a_n \in A$ tal que toda permutación de A que fija a cada a_i con $1 \leq i \leq n$

es un automorfismo de $\mathcal{A}$.

Observación. Toda estructura con dominio finito es trivial.

Demostración. Podemos pedir que se fije a toda la estructura y así tener que implicar que toda permutación que fije a la estructura debe ser la identidad, la cual es un automorfismo. □

Una característica de las estructuras triviales es que todas sus ω-presentaciones tienen las mismas propiedades de computabilidad. Más precisamente se satisface lo siguiente.

Teorema 5.1.1. *Si $\mathcal{A}$ es una $\mathcal{L}$-estructura trivial entonces todas sus ω-presentaciones son isomorfas mediante biyecciones computables.*

Demostración. Sean $\mathcal{M}_1$, $\mathcal{M}_2$ dos ω-presentaciones de $\mathcal{A}$ y $f : \omega \to \omega$ y $g : \omega \to \omega$ isomorfismos entre $\mathcal{M}_1$ y $\mathcal{A}$ y entre $\mathcal{A}$ y $\mathcal{M}_2$ respectivamente. Observemos que $f \circ g$ es un isomorfismo entre $\mathcal{M}_1$ y $\mathcal{M}_2$, pero del cual no tenemos certeza si es computable. Por ello daremos una construcción de un isomorfismo computable entre $\mathcal{M}_1$ y $\mathcal{M}_2$.

Como $\mathcal{A}$ es trivial, existen $a_1, ..., a_n \in A$ tales que toda permutación de A que fija dichos elementos es un automorfismo. Definimos los conjuntos $R := \{r_1, ..., r_n\}, S := \{s_1, ..., s_n\} \subseteq \omega$ tales que $f(r_i) = a_i$ y $g(a_i) = s_i$ para cada $1 \leq i \leq n$.

Si $R = S$, definimos $h : \omega \to \omega$ tal que

$$h(m) = \begin{cases} a_i & \text{si } m = a_i \text{ para algún } 1 \leq i \leq n \\ g^{-1}(f^{-1}(m)) & \text{si ocurre otro caso} \end{cases}$$

Veamos que h es inyectiva. En efecto supongamos que $h(m) = h(m')$ con $m, m' \in \omega$.

En caso de que m, m' sean tales que $m = a_i$ y $m' = a_j$ para algunos i, j con $1 \leq i, j \leq n$, entonces por la definición de h, $a_i = a_j$, por lo tanto, $m = m'$.

Si $m = a_i$ y $m' \neq a_k$ para ningún k entonces $a_i = g^{-1}(f^{-1}(m'))$. Luego $g(a_i) = f^{-1}(m')$, lo cual implica que $f^{-1}(m') \in S$. Como $R = S$, $f^{-1}(m') = r_j$ para algún $1 \leq j \leq n$. Luego, $m' = f(f^{-1}(m')) = f(r_i) = a_j$, lo cual es una contradicción. Por lo tanto, este caso no puede ocurrir.

Si $m \neq a_i$ y $m' \neq a_j$ para ningunos i, j entonces $g^{-1}(f^{-1}(m)) = g^{-1}(f^{-1}(m'))$, luego como g^{-1} y f^{-1} son, en particular, inyectivas, tenemos que $m = m'$. Por lo tanto, concluimos que h es inyectiva.

Veamos que h es sobreyectiva. Sea $k \in \omega$ y veamos que existe $m \in \omega$ tal qeu $h(m) = k$.

En efecto, si $k = a_i$ para algún i entonces definimos $m = a_i$ y tenemos que $h(m) = h(a_i) = a_i = k$.

Si $k \neq a_i$ para ningún i, definimos $m = f(g(k))$. Entonces $h(m) = g^{-1}(f^{-1}(m)) = g^{-1}(f^{-1}(f(g(k)))) = k$. Por lo tanto, concluimos que h es sobreyectiva.

Por lo tanto, es biyectiva. Por la definición de h, esta fija a cada a_i y es una permutación en ω. Como $\mathcal{A}$ es trivial, h es un automorfismo de $\mathcal{A}$.

Definimos $F : \omega \to \omega$ tal que $F = g \circ h \circ f$, entonces F es un isomorfismo entre M_1 y M_2.

Veamos que F es computable. Sea $m \in \omega$ tal que $m \notin R$. Entonces $f(m) \neq a_i$ para ningún i, luego $h(f(m)) = g^{-1}(f^{-1}(f(m))) = g^{-1}(m)$. Entonces podemos implicar que $F(m) = g(h(f(m))) = g(g^{-1}(m)) = m$. Definimos un programa que guarde los valores de $F(m)$ para cada $m \in R$. Como R es finito la cantidad de estos valores es finita. Y que dicho programa en un elemento $m \notin R$ que regrese el mismo valor m. Es evidente que la función inducida por dicho programa es F. Por lo tanto, F es computable. En este caso concluimos que F es un isomorfismo computable entre $\mathcal{M}_1$ y $\mathcal{M}_2$.

Si $R \neq S$ entonces $R \setminus S \neq \emptyset$ ó $S \setminus R \neq \emptyset$. Supongamos que $R \setminus S \neq \emptyset$. Si $S \setminus R = \emptyset$ entonces $S \subseteq R$. Como la cardinalidad de S y R es la misma entonces $S = R$, lo cual es una contradicción. Por lo tanto, $S \setminus R \neq \emptyset$. De la misma forma, si suponemos que $S \setminus R \neq \emptyset$ entonces podemos concluir que $R \setminus S \neq \emptyset$. Por lo tanto, $R \setminus S \neq \emptyset$ y $S \setminus R \neq \emptyset$.

Notemos que $|R| = |R \setminus S| + |R \cap S|$ y $|S| = |S \setminus R| + |S \cap R|$. Por lo tanto $|R \setminus S| = |S \setminus R|$. Sea $f_* : S \setminus R \to R \setminus S$ una biyección.

Definimos $h : \omega \to \omega$ tal que:

$$h(m) = \begin{cases} a_i & \text{si } m = a_i \text{ para algún } 1 \leq i \leq n \\ g^{-1}(f_*(f^{-1}(m))) & \text{si } f^{-1}(m) \in S \setminus R \\ g^{-1}(f^{-1}(m)) & \text{si } f^{-1}(m) \in \omega \setminus (S \cup R) \end{cases}$$

Veamos que h es inyectiva. En efecto, sean $m, m' \in A$ tales que $h(m) = h(m')$.

Si $a = a_i, a' = a_j$ para algunos i, j entonces $m = a_i = h(m) = h(m') = a_j = m'$.

Si $m = a_i$ y $m' \neq a_i$ para ningún i entonces $f^{-1}(m') \notin R$. Si $f^{-1}(m') \in S$ entonces $a_i = h(m) = h(m') = g^{-1}(f_*(f^{-1}(m')))$. Por lo tanto, $g(a_i) = f_*(f^{-1}(m')) \in R \setminus S$,

es decir $g(a_i) \notin S$, lo cual es una contradicción por la definición de S.

Si $f^{-1}(m') \notin S$ entonces $a_i = h(m) = h(m') = g^{-1}(f^{-1}(m'))$, luego $g(a_i) = f^{-1}(m') \notin S$, lo cual nuevamente es una contradicción por la definición de S. Por consiguiente este caso no puede ocurrir. De manera similar podemos ver que el caso $m \neq a_i$ para ningún i y $m' = a_i$ para algún i no sucede.

Si $m \neq a_i, m' \neq a_j$ para ningunos i, j entonces $f^{-1}(m), f^{-1}(m') \notin R$. Si $f^{-1}(m), f^{-1}(m') \in S$ entonces $g^{-1}(f_*(f^{-1}(m))) = h(m) = h(m') = g^{-1}(f_*(f^{-1}(m')))$. Como g^{-1}, f_*, f^{-1} son biyectivas entonces $m = m'$.

Si $f^{-1}(m) \in S$ pero $f^{-1}(m') \notin S$ entonces $g^{-1}(f_*(f^{-1}(m))) = h(m) = h(m') = g^{-1}(f^{-1}(m'))$. Por lo tanto, $f_*(f^{-1}(m)) = f^{-1}(m') \notin R$, lo cual contradice la definición de f_*. Concluimos que este caso no puede ocurrir. De la misma forma obtenemos que $f^{-1}(m) \notin S$ y $f^{-1}(m') \in S$ no puede ocurrir.

Si $f^{-1}(m), f^{-1}(m') \notin S$ entonces $g^{-1}(f^{-1}(m)) = h(m) = h(m') = g^{-1}(f^{-1}(m'))$, lo cual implica que $m = m'$. Concluimos finalmente que h es inyectiva.

Veamos que h es sobreyectiva. Sea $k \in \omega$. Si $k = a_i$ para algún i entonces definimos $m = a_i$. Por lo tanto, $h(m) = a_i = k$.

Si $k \neq a_i$ para ningún i distingamos de dos casos.

Si $g(k) \in R$ entonces definimos $m = f(f_*^{-1}(g(k)))$. Observemos que $f^{-1}(m) \in S \setminus R$. Por lo tanto, $h(m) = g^{-1}(f_*(f^{-1}(m))) = g^{-1}(f_*(f^{-1}(f(f_*^{-1}(g(k)))))) = k$.

Ahora si $g(k) \notin R$ entonces definimos $m = f(g(k))$. Observemos que $f^{-1}(m) \in \omega \setminus (S \cup R)$. Por lo tanto, $h(m) = g^{-1}(f^{-1}(m)) = g^{-1}(f^{-1}(f(g(k)))) = k$. Concluimos que h es sobreyectiva y por lo tanto que h es biyectiva. Entonces h es una permutación en ω que fija a cada a_i, por lo tanto concluimos que h es un automorfismo de $\mathcal{A}$.

Definimos $F : \omega \to \omega$ tal que $F = g \circ h \circ f$. Notemos que F es un isomorfismo entre $\mathcal{M}_1$ y $\mathcal{M}_2$. Veamos que F es computable, para ello notemos que si $m \in \omega \setminus (R \cup S)$ entonces $F(m) = (g \circ h \circ f)(m) = g(h(f(m))) = g(g^{-1}(f^{-1}(f(m)))) = m$.

Luego, como $R \cup S$ es finito, podemos guardar en la memoria de un programa los valores de $F(m)$ para cada $m \in R \cup S$ y para elementos $m \in \omega \setminus (R \cup S)$ pedimos que el programa devuelva el mismo valor. La función inducida de este programa es F. Por lo tanto F es computable.

En cualquier caso, hemos construido un isomorfismo computable entre $\mathcal{M}_1$ y $\mathcal{M}_2$. Por lo tanto, cualesquiera ω-presentaciones de una estructura trivial son isomorfas mediante biyecciones computables.

□

Corolario 5.1.1. *Si $\mathcal{A}$ es una $\mathcal{L}$-estructura trivial entonces todas sus ω-presentaciones tienen el mismo grado de Turing.*

Demostración. Sean $\mathcal{M}_1, \mathcal{M}_2$ dos ω-presentaciones de $\mathcal{A}$. Recordemos que podemos

suponer sin pérdida de generalidad que $\mathcal{M}_1$ y $\mathcal{M}_2$ son relacionales. Ahora bien por el teorema anterior, existe un isomorfismo computable f entre $\mathcal{M}_1$ y $\mathcal{M}_2$. Veamos que $D(\mathcal{M}_1) \leq_T D(\mathcal{M}_2)$. Sea φ_i^{at} una fórmula atómica en el lenguaje de $\mathcal{M}_1$, que es el mismo lenguaje de $\mathcal{M}_2$. Supongamos que φ_i^{at} es de la forma $R_j(x_{k_1}, ..., x_{k_{a_R(j)}})$ para algún $j \in \omega$. Definimos un programa que de forma computable obtenga el índice de la fórmula atómica $R_j(x_{f(k_1)}, ..., x_{f(k_{a_R(j)})})$, digamos $t(i)$.

Observemos que $D(\mathcal{M}_1)(i) = 1$ si y sólo si $(k_1, ..., k_{a_R(j)}) \in R_j^{\mathcal{M}_1}$, lo cual equivale a que $(f(k_1), ..., f(k_{a_R(j)})) \in R_j^{\mathcal{M}_2}$, que ocurre si y sólo si $D(\mathcal{M}_2)(t(i)) = 1$. Luego, $D(\mathcal{M}_1)(i) = D(\mathcal{M}_2)(t(i))$.

Finalmente hacemos que el programa devuelva $D(\mathcal{M}_2)(t(i))$, es decir $D(\mathcal{M}_1)(i)$. La función definida por dicho programa es $D(\mathcal{M}_1)$. Por lo tanto $D(\mathcal{M}_1)$ es $D(\mathcal{M}_2)$-computable, es decir $D(\mathcal{M}_1) \leq_T D(\mathcal{M}_2)$. Usando que f^{-1} es un isomorfismo computable entre $\mathcal{M}_2$ y $\mathcal{M}_1$, de forma similar podemos concluir que $D(\mathcal{M}_2) \leq_T D(\mathcal{M}_1)$.

Concluimos que $D(\mathcal{M}_1) \equiv_T D(\mathcal{M}_2)$, es decir que $\mathcal{M}_1$ y $\mathcal{M}_2$ tienen el mismo grado de Turing. Por consiguiente todas las ω-presentaciones de $\mathcal{A}$ tienen el mismo grado de Turing.

□

El corolario anterior esencialmente nos dice que si queremos una estructura con presentaciones con distintos grados de Turing, las estructuras triviales no son el lugar adecuado para buscar. A continuación veremos que la condición de no trivialidad basta para obtener estructuras con presentaciones de distintos grados de Turing.

Teorema 5.1.2. *Sean $X \subseteq \omega$ y $\mathcal{M}$ una ω-presentación de una $\mathcal{L}$-estructura no trivial $\mathcal{A}$. Si $\mathcal{M}$ es X-computable entonces existe una ω-presentación de $\mathcal{A}$ del mismo grado de Turing que X.*

Demostración. La idea de la demostración es construir una función $g : \omega \to \omega$ biyectiva y X-computable, para así poder definir la ω-presentación que buscamos como el pullback de $\mathcal{M}$ inducido por g.

Construiremos recursivamente a g mediante una sucesión de funciones parciales finitas $g_s : \text{dom } g_s \subseteq \omega \to \omega$ tal que $g_s \subseteq g_{s+1}$ para cada $s \in \omega$ y tal que tras tomar la unión de todas ellas la función resultante sea biyectiva X-computable.

Como breve paréntesis, en ocasiones a fin de ahorrarnos notación escribiremos únicamente g_s para denotar la sucesión finita $g_s(0), ..., g_s(|\text{dom } g_s| - 1)$.

Comencemos con la construcción definiendo $g_0 = \emptyset$. Supongamos que hemos definido g_s y definamos g_{s+1}.

Para ello buscamos tuplas $\overline{a}, \overline{b} \in (\omega \setminus \{g_s(0), ..., g_s(|\text{dom } g_s| - 1)\})^{<\mathbb{N}}$ tal que $D_{\mathcal{M}}(g_s\overline{a}) \neq D_{\mathcal{M}}(g_s\overline{b})$. Como $\mathcal{M}$ no es trivial, para la tupla g_s existe una permutación de ω, digamos f_s, que fija a g_s pero que no es un automorfismo. Dado que $\mathcal{L}$ es un lenguaje relacional, debe ocurrir que existen $j, k_1, ..., k_{a_R(j)} \in \omega$ tal que $(k_1, ..., k_{a_R(j)}) \in R_j^{\mathcal{M}}$ pero $(f_s(k_1), ..., f_s(k_{a_R(j)})) \notin R_j^{\mathcal{M}}$ o bien que $(f_s(k_1), ..., f_s(k_{a_R(j)})) \in R_j^{\mathcal{M}}$ pero $(k_1, ..., k_{a_R(j)}) \notin R_j^{\mathcal{M}}$.

Supongamos que $(k_1, ..., k_{a_R(j)}) \in R_j^{\mathcal{M}}$ pero $(f_s(k_1), ..., f_s(k_{a_R(j)})) \notin R_j^{\mathcal{M}}$ (el otro caso es similar).

Como f_s fija a la tupla g_s entonces no puede ocurrir que todos los k_i sean elementos de la tupla g_s, luego existen $k_{i_0}, ..., k_{i_n}$ todos distintos entre sí y tal que $i_0 < \cdots < i_n$ que no pertenecen a g_s. Por otro lado como cada k_{i_l} no está en la tupla g_s entonces $f_s(k_{i_l})$ tampoco está en la tupla g_s ya que f_s es inyectiva y fija a la tupla g_s.

Sean $r_{i_0}, ..., r_{i_n}$ elementos en ω tales que $r_{i_0} < ... < r_{i_n}$ (son distintos entre sí), distintos de cada elemento $k_{i_0}, ..., k_{i_n}$, de $f_s(k_{i_0}), ..., f_s(k_{i_n})$, tales que ninguno está en la tupla g_s y tal que $(r_{i_0}, ..., r_{i_n})$ es la menor tupla respecto al orden lexicográfico que cumple esta propiedad. Sea k^* la tupla formada a partir de sustituir cada k_{i_l} por r_{i_l} en la tupla $(k_1, ..., k_{a_R(j)})$. Distingamos de dos casos: si $k^* \in R_j^{\mathcal{M}}$ entonces definimos $\overline{c} = (c_0, ..., c_n) = (r_{i_0}, ..., r_{i_n})$ y $\overline{d} = (d_0, ..., d_n) = (f_s(k_{i_0}), ..., f_s(k_{i_n}))$; si $k^* \notin R_j^{\mathcal{M}}$ entonces definimos $\overline{c} = (c_0, ..., c_n) = (k_{i_0}, ..., k_{i_n})$ y $\overline{d} = (d_0, ..., d_n) = (r_{i_0}, ..., r_{i_n})$. Notemos que las tuplas $\overline{c}$ y $\overline{d}$ no comparten ningún elemento en común y además no tienen elementos en común con la tupla g_s.

Notemos lo siguiente: si quisiéramos calcular el diagrama atómico de una tupla en $\mathcal{M}$ y al mismo tiempo quisiéramos que éste considerara a fórmulas atómicas cuyo símbolo relacional sea R_j entonces debemos asegurar que la longitud de la tupla en cuestión sea mayor o igual a j.

Para poder resolver esta problemática realizamos el siguiente proceso. Primero utilizando $D(\mathcal{M})$, vemos si $D_{\mathcal{M}}(g_s c_0) \neq D_{\mathcal{M}}(g_s d_0)$; si esto ocurre definimos $\overline{a} = c_0$ y $\overline{b} = d_0$. En caso contrario vemos ahora si $D_{\mathcal{M}}(g_s c_0 c_1) \neq D_{\mathcal{M}}(g_s d_0 d_1)$; si esto ocurre definimos $\overline{a} = c_0 c_1$ y $\overline{b} = d_0 d_1$, en caso contrario continuamos el proceso. En caso de que $D_{\mathcal{M}}(g_s c_0 c_1 ... c_{n-1}) = D_{\mathcal{M}}(g_s d_0 d_1 ..., d_{n-1})$ distinguimos de dos casos:

(a) Si $j \leq |g_s| + n$, sea t el índice de la fórmula que cumple que al sustituir cada variable de esta por el elemento de la tupla $g_s\overline{c}$ cuya posición es determinada por el índice de dicha variable da como resultado la tupla $(k_1, ..., k_{a_R(j)})$ ó la tuplak^* y cuando la sustitución es respecto a la tupla $g_s\overline{d}$ el resultado es k^* ó la tupla $(f_s(k_1), ..., f_s(k_{a_R(j)}))$. Entonces se cumple que $D_{\mathcal{M}}(g_s c_0 c_1 ... c_{n-1} c_n)(t) \neq D_{\mathcal{M}}(g_s d_0 d_1 ... d_{n-1} d_n)(t)$, por lo tanto

$D_{\mathcal{M}}(g_s c_0 c_1 ... c_{n-1} c_n) \neq D_{\mathcal{M}}(g_s d_0 d_1 ... d_{n-1} d_n)$. En este caso definimos $\overline{a} = \overline{c}$ y $\overline{b} = \overline{d}$.

(b) Si $|g_s| + n < j$ entonces tomamos dos elementos distintos entre sí $u_0, v_0 \in \omega$, distintos de los elementos de las tuplas $g_s\overline{c}$ y $g_s\overline{d}$ y vemos si $D_{\mathcal{M}}(g_s c_0 .... c_{n-1} u_0) = D_{\mathcal{M}}(g_s d_0 ... d_{n-1} v_0)$ y de forma similar a como hemos hecho anteriormente en esta construcción si esto no ocurre definimos $\overline{a} = c_0 ... c_{n-1} u_0$ y $\overline{b} = d_0 ... d_{n-1} v_0$.

En caso de que la igualdad se cumpla verificamos si $|g_s| + n + 1 < j$. Si $|g_s| + n + 1 = j$, entonces debemos obtener que $D_{\mathcal{M}}(g_s c_0 ... c_{n-1} u_0 c_n) \neq D_{\mathcal{M}}(g_s d_0 ... d_{n-1} v_0 d_n)$ (por una razón similar al caso en el que $j \leq |g_s| + n$) entonces elegimos $\overline{a} = c_0 ... c_{n-1} u_0 c_n$ y $\overline{b} = d_0 ... d_{n-1} v_0 d_n$.

Si $|g_s| + n + 1 < j$, buscamos ahora elementos distintos entre sí $u_1, v_1 \in \omega$ distintos de los elementos de las tuplas $g_s\overline{c}u_0$ y $g_s\overline{d}v_0$. Nuevamente verificamos si $D_{\mathcal{M}}(g_s c_0 .... c_{n-1} u_0 u_1) = D_{\mathcal{M}}(g_s d_0 ... d_{n-1} v_0 v_1)$ y si esto ocurriera preguntamos si $|g_s| + a_R(j) + 2 < j$ y continuamos con el proceso de forma similar. Observemos que en algún momento este proceso se detiene. Por consiguiente, tenemos finalmente tuplas $\overline{a} = (a_0, ..., a_{h-1})$ y $\overline{b} = (b_0, ..., b_{h-1})$ cuyos elementos todos son distintos entre sí tales que $D_{\mathcal{M}}(g_s a_0 ... a_{i-1}) = D_{\mathcal{M}}(b_0 .... b_{i-1})$ para cada $i < h$ y $D_{\mathcal{M}}(g_s\overline{a}) \neq D_{\mathcal{M}}(g_s\overline{b})$.

Ahora, supongamos sin pérdida de generalidad que $D_{\mathcal{M}}(g_s\overline{a}) <_{lex} D_M(g_s\overline{b})$ donde $\leq_{lex}$ es el orden lexicográfico de tuplas finitas de ω (en caso contrario cambiamos $\overline{a}$ por $\overline{b}$ y $\overline{b}$ por $\overline{a}$). Si $\chi_X(s) = 0$ definimos $g_s^* = g_s a_0 b_0 ... a_{h-1} b_{h-1}$ y en caso de que $\chi_X(s) = 1$ entonces definimos $g_s^* = g_s b_0 a_0 ... b_{h-1} a_{h-1}$.
Finalmente, sea r el ω-menor elemento de ω distinto de todos los elementos de g_s^* y definimos $g_{s+1} = g_s^* r$.

Es evidente que $g_s \subseteq g_{s+1}$. Por consiguiente $g := \bigcup_{s\in\omega} g_s$ es una función. Como en cada etapa de la construcción de cada g_s se agrega al menos un elemento tenemos que dom $g = \omega$ y como g devuelve números naturales entonces el contradominio de g es ω.

Más aún como para cada $s \in \omega$, $s \in g_{s+1}$ entonces la imagen de g_{s+1} contiene a s. Como g es la unión de todas las g_s entonces g es sobreyectiva.

Observemos que en cada etapa de la construcción de g_s nos aseguramos que los elementos que agregaramos fueran todos distintos entre sí y además que fueran todos distintos de los que ya se habían agregado en etapas de construcción anteriores. Por lo tanto podemos concluir que g es una función inyectiva. En conclusión g es una función biyectiva.

Notemos también que en cada etapa para obtener a la función parcial g_s vista como sucesión hicimos uso del diagrama atómico $D(\mathcal{M})$ y del conjunto X. Como $\mathcal{M}$ es X-computable, entonces podemos concluir que para obtener los valores de g_s podemos utilizar únicamente al conjunto X. Nuevamente como g es la unión de todas las g_s podemos concluir que para obtener los valores de g basta con utilizar al conjunto X. Por lo tanto g es X-computable.

Sea $\mathcal{N} = g^{-1}(\mathcal{M})$. Como g es biyectiva, $\mathcal{N}$ es una ω-presentación de $\mathcal{A}$. Por la proposición 4,2,4 tenemos que $D(\mathcal{N}) \leq_T g \oplus D(\mathcal{M})$. Como g y $\mathcal{M}$ son X-computables entonces $g \oplus D(\mathcal{M}) \leq_T X$. Por lo tanto $D(\mathcal{N}) \leq_T X$.

Veamos que $X \leq_T D(\mathcal{N})$. Para cada $s \in \omega$ sea $q_s = |g_s|$. En la construcción del paso $s+1$ notemos que q_{s+1} es el menor $q > q_s$ tal que $D_{\mathcal{M}}(g(0), ..., g(q_s - 1), g(q_s), g(q_s+2), ..., g(q-3)) \neq D_{\mathcal{M}}(g(0), ..., g(q_s-1), g(q_s+1), g(q_s+3), ..., g(q-2))$.

Como

$$D_{\mathcal{M}}(g(0), ..., g(q_s-1), g(q_s), g(q_s+2), ..., g(q-3)) = D_{\mathcal{N}}(0, ..., q_s-1, q_s, q_s+2, ..., q-3)$$

y

$$D_{\mathcal{M}}(g(0), ..., g(q_s-1), g(q_s+1), g(q_s+3), ..., g(q-2)) = D_{\mathcal{N}}(0, ..., q_s-1, q_s+1, ..., q-2)$$

entonces q_{s+1} es el menor $q > q_s$ tal que

$$D_{\mathcal{N}}(0, ..., q_s - 1, q_s, q_s + 2, q_s + 4, ..., q - 3) \neq D_{\mathcal{N}}(0, ..., q_s - 1, q_s + 1, q_s + 3, ..., q - 2)$$

De este modo podemos obtener de manera sucesiva los valores q_s utilizando a $D(\mathcal{N})$ como oráculo. Con esto podemos definir un programa que utilice a $D(\mathcal{N})$ como oráculo, que obtenga de forma sucesiva los valores de cada q_i para cada $i \leq s+1$ y así pueda obtener los valores de los diagramas atómicos $D_{\mathcal{N}}(0, ..., q_s - 1, q_s, q_s + 2, q_s + 4, ..., q_{s+1} - 3)$ y $D_{\mathcal{N}}(0, ..., q_s - 1, q_s + 1, q_s + 3, ..., q_{s+1} - 2)$ para poder compararlos respecto al orden lexicográfico. Si obtenemos que $D_{\mathcal{N}}(0, ..., q_s - 1, q_s, q_s + 2, q_s + 4, ..., q_{s+1} - 3)$ es menor entonces $\chi_X(s) = 0$ por lo que el programa devuelve 0 y termina. En otro caso devuelve 1. Tenemos que la función definida por este programa es χ_X, lo cual implica que X es $D(\mathcal{N})$-computable, es decir que $X \leq_T D(\mathcal{N})$. Esto nos permite concluir que $D(\mathcal{N}) \equiv_T X$.

□

Algunos resultados del teorema anterior son los siguientes.

Corolario 5.1.2. *Si $\mathcal{A}$ es una estructura no trivial con una ω-presentación computable entonces $\mathcal{A}$ tiene una ω-presentación con grado de Turing X para cualquier conjunto $X \subseteq \omega$.*

Demostración. Sean $X \subseteq \omega$ y $\mathcal{M}$ una ω-presentación de $\mathcal{A}$ que es computable. Entonces $\mathcal{M}$ es X-computable. Por el teorema anterior existe una ω-presentación de $\mathcal{A}$ cuyo grado de Turing es X. □

Ejemplo 5.1.1. Sea $\mathcal{N} = (\omega; +, \times, S, 0, 1)$ la estructura donde $+^{\mathcal{N}}$, $\times^{\mathcal{N}}$, $S^{\mathcal{N}}$ son la suma, multiplicación y operación sucesor usuales de los números naturales y $0^{\mathcal{N}}$, $1^{\mathcal{N}}$ son los números 0 y 1 respectivamente. A esta estructura la llamaremos **modelo estándar de la aritmética**. Entonces para cualquier conjunto $X \subseteq \omega$, $\mathcal{N}$ tiene una ω-presentación de grado de Turing X.

Demostración. Se sigue del corolario anterior y de que $\mathcal{N}$ es una ω-presentación de sí misma que es computable. □

Como no todas las ω-presentaciones de una estructura cualquiera tienen el mismo grado de Turing definimos el siguiente conjunto.

Definición 5.1.2. (Espectro de Grados) Sea $\mathcal{A}$ una estructura. Definimos al **espectro de grados** de $\mathcal{A}$ como el conjunto

$$DgSp(\mathcal{A}) = \{X \subseteq \omega : X \text{ cómputa una } \omega\text{-presentación de } \mathcal{A}\}$$

Un par de observaciones inmediatas son las siguientes.

Observación. Notemos que si $\mathcal{A}$ y $\mathcal{B}$ son dos estructuras isomorfas entonces $DgSp(\mathcal{A}) = DgSp(\mathcal{B})$. En particular, si $\mathcal{A}$ es una estructura, no importa cuál de sus ω-presentaciones tomemos, si calculamos su espectro de grados siempre tendremos el mismo de $\mathcal{A}$.

Demostración. Se sigue de la transitividad de la propiedad ser isomorfos. □

Observación. Sean $\mathcal{A}$ es una estructura y $X, Y \subseteq \omega$. Si $X \in DgSp(\mathcal{A})$ y $X \leq_T Y$ entonces $Y \in DgSp(\mathcal{A})$. En particular, si $X \equiv_T Y$ entonces $Y \in DgSp(\mathcal{A})$.

Demostración. Se sigue de la transitividad de la relación $\leq_T$. □

Una consecuencia de la observación anterior es que podemos considerar al espectro de grados como un conjunto de grados de Turing. Si la estructura no es trivial, por el teorema 5.1.2 podemos ver su espectro de grados como el conjunto de grados de Turing de sus ω-presentaciones.

5.2. Computabilidad de isomorfismos entre ω-presentaciones de una estructura (versión uniforme)

Ahora abordaremos el hecho de que dos ω-presentaciones de una estructura sean isomorfas no implica que exista un isomorfismo computable entre ellas. Veamos el siguiente ejemplo.

Ejemplo 5.2.1. Sea $\mathcal{A} = (\omega; \leq)$ la estructura donde $\leq^{\mathcal{A}}$ se interpreta como la desigualdad usual en los números naturales. Como breve paréntesis, en ocasiones eliminaremos el superíndice $\mathcal{A}$ de $\leq^{\mathcal{A}}$ para únicamente escribir $\leq$. Sea $f : \omega \to \omega$ una función computable e inyectiva tal que im $f = K$ donde $K = \{e \in \omega : \Phi_e(e) \downarrow\}$. Denotamos por $k_n := f(n)$. Sea $\mathcal{M} = (\omega; \leq)$ la estructura donde $\leq^{\mathcal{M}}$ es tal que:

(a) $2n \leq^{\mathcal{M}} 2m$ si y sólo si $n \leq^{\mathcal{A}} m$

(b) $2k_n \leq^{\mathcal{M}} 2n+1 \leq^{\mathcal{M}} 2k_n+2$

(c) $2n \leq^{\mathcal{M}} 2m+1$ si y sólo si $n \leq^{\mathcal{A}} k_m$

(d) $2m+1 \leq^{\mathcal{M}} 2n$ si y sólo si $k_m+1 \leq^{\mathcal{A}} n$

(e) $2m+1 \leq^{\mathcal{M}} 2n+1$ si y sólo si $k_m \leq^{\mathcal{A}} k_n$

Entonces $\mathcal{A}$ y $\mathcal{M}$ son isomorfas y computables pero no existe un isomorfismo computable entre ellas.

Demostración. Hagamos las siguientes observaciones de $\leq^{\mathcal{M}}$.

(i) Para cualesquiera $n, m \in \omega$ si $n \leq^{\mathcal{M}} m$ y $m \leq^{\mathcal{M}} n$ entonces $n = m$.

(ii) Para cualesquiera $n, m \in \omega$, $n \leq^{\mathcal{M}} m$ ó $m \leq^{\mathcal{M}} n$.

(iii) Todo $m \in \omega$ tiene una cantidad finita de $\leq^{\mathcal{M}}$-predecesores.

Demostremos el inciso *(i)*. Sean $n, m \in \omega$ tales que $n \leq^{\mathcal{M}} m$ y $m \leq^{\mathcal{M}} n$. Distingamos de los siguientes casos:

> Supongamos que $n = 2j$ para algún $j \in \omega$, es decir que n es par. Si $m = 2i+1$ para algún $i \in \omega$, es decir que m es impar entonces $j \leq k_i$ y $k_i + 1 \leq j$, lo cual es una contradicción. Por lo tanto este caso no sucede. Entonces debe ocurrir que $m = 2i$ para algún $i \in \omega$, es decir que m. Entonces $j \leq i$ y $i \leq j$. Esto nos permite concluir que $n = m$.
>
> Ahora bien si $n = 2j+1$ par algún $j \in \omega$, es decir que n es impar. Si $m = 2i$ para algún $i \in \omega$, es decir que m es par entonces $k_i + 1 \leq j$ y $j \leq k_i$, lo cual es una contradicción. Por lo tanto este caso no sucede. Entonces $m = 2i+1$ para algún $i \in \omega$. Luego $k_j \leq k_i$ y $k_i \leq k_j$, entonces $k_i = k_j$. Como f es inyectiva tenemos que $i = j$, por lo tanto $n = m$.

Con esto queda demostrado el inciso *(i)*.

Demostremos ahora el inciso *(ii)*. Sean $n, m \in \omega$.

> Supongamos que $n = 2j$ para algún $j \in \omega$, es decir que n es par. Distingamos de dos casos:

(a) Si $m = 2i+1$ para algún $i \in \omega$, es decir que m es impar entonces tenemos que o bien $j \leq k_i$ ó $k_i + 1 \leq j$. La primera desigualdad nos implica que $2j \leq^{\mathcal{M}} 2i + 1$, es decir $n \leq^{\mathcal{M}} m$. La segunda que $2i + 1 \leq^{\mathcal{M}} 2j$, es decir $m \leq^{\mathcal{M}} n$.

(b) Ahora bien si $m = 2i$ para algún $i \in \omega$, es decir que m es par entonces tenemos que $j \leq i$ ó $i \leq j$. La primera desigualdad implica que $2j \leq^{\mathcal{M}} 2i$, es decir que $n \leq^{\mathcal{M}} m$. De la segunda obtenemos que $m \leq^{\mathcal{M}} n$.

Si $n = 2j + 1$ par algún $j \in \omega$, es decir n es impar. Distinguimos de dos casos:

(a) Si $m = 2i+1$ para algún $i \in \omega$, es decir que m es impar entonces tenemos que o bien $k_j \leq k_i$ ó $k_i \leq k_j$. Es sencillo ver que obtenemos que $n \leq^{M} m$ ó $m \leq^{\mathcal{M}} n$.

(b) Ahora bien si $m = 2i$ para algún $i \in \omega$, es decir que m es par entonces el caso *(a)* nos dice que $m \leq^{\mathcal{M}} n$ ó $n \leq^{\mathcal{M}} m$.

Con lo cual queda demostrado el inciso *(ii)*.

Ahora demostremos el inciso *(iii)*. Sea $m \in \omega$. Definimos como $Par_m := \{n \in \omega : n \leq^{\mathcal{M}} m \wedge n \text{ es par}\}$ al conjunto de $\leq^{\mathcal{M}}$-predecesores pares de m e $Impar_m := \{n \in \omega : n \leq^{\mathcal{M}} m \wedge n \text{ es impar}\}$ al conjunto de $\leq^{\mathcal{M}}$-predecesores impares de m. Observemos que $Par_m \cup Impar_m$ es el conjunto de $\leq^{\mathcal{M}}$-predecesores de m.

Supongamos que $m = 2s$ para algún $s \in \omega$.

Definimos la función $h_m : Par_m \to \omega$ tal que $h_m(n) = j$ si $n = 2j$ para algún $j \in \omega$.

Veamos que $h_m[Par_m] \subseteq \{0, 1, ..., s\}$. Sea $n \in Par_m$, digamos de la forma $2j$, entonces $h_m(n) = j$. Como $n \in Par_m$, $n \leq^{M} m$. Por lo tanto, $2j \leq^{\mathcal{M}} 2s$, entonces $j \leq^{\mathcal{A}} s$, es decir $h_m(n) \leq^{\mathcal{A}} s$. Por consiguiente $h_m(n) \in \{0, 1, ..., s\}$.

Ahora veamos que Par_m es finito. Para ello veamos que h_m es inyectiva. En efecto, si $n, l \in Par_m$ son tales que $n = 2j$ y $l = 2i$ para algunos $i, j \in \omega$ entonces $h_m(n) = h_m(l)$ implica que $j = i$. Por lo tanto $n = l$. Concluimos que h_m es inyectiva. Más aún, tenemos que $|Par_m| = |h_m[Par_m]| \leq s + 1$. Por lo tanto Par_m es finito.

Definimos la función $t_m : Impar_m \to \omega$ tal que $t_m(n) = k_j + 1$ si $n = 2j + 1$ para algún $j \in \omega$.

Veamos que $t_m[Impar_m] \subseteq \{0, 1, ..., s\}$. Sea $n \in Impar_m$ de la forma $2j + 1$ para algún $j \in \omega$. Entonces $t_m(n) = k_j + 1$. Como $n \in Impar_m$ entonces $n \leq^{\mathcal{M}} m$. Por consiguiente $2j + 1 \leq^{\mathcal{M}} 2s$, lo cual implica que $k_j + 1 \leq^{\mathcal{A}} s$. Por lo tanto,

$k_j + 1 \in \{0, 1, ..., s\}$, lo cual nos implica que $t_m(n) \in \{0, 1, ..., s\}$. Concluimos que $t_m[Impar_m] \subseteq \{0, 1, ..., s\}$.

Ahora veamos que t_m es inyectiva. Sean $n, l \in Impar_m$ tales que $n = 2j + 1$ y $l = 2i + 1$ y tales que $t_m(n) = t_m(l)$. Entonces tenemos que $k_j + 1 = k_i + 1$, por lo tanto $k_j = k_i$. Por la inyectividad de la función enumeración f tenemos que $j = i$ y por lo tanto podemos concluir que $n = l$. Entonces t_m es inyectiva. Entonces $|Impar_m| = |t_m[Impar_m]| \leq s + 1$. Concluimos que $Impar_m$ es finito.

Por lo tanto el conjunto de $\leq^{\mathcal{M}}$-predecesores de m es finito.

De manera análoga podemos concluir que si $m = 2s + 1$ para algún $s \in \omega$ entonces el conjunto de $\leq^{\mathcal{M}}$-predecesores de m es finito. Esto demuestra el inciso *(iii)*.

Con ayuda de las observaciones anteriores veamos que $\mathcal{A}$ y $\mathcal{M}$ son isomorfas. Definamos un isomorfismo de $\mathcal{A}$ a $\mathcal{M}$ recursivamente. Definimos $g_0 : \{0\} \to \omega$ tal que $g_0(0) = 0$. Supongamos que hemos definido $g_s : \{0, 1, ..., s\} \to \omega$. Definimos $g_{s+1} : \{0, 1, ..., s + 1\} \to \omega$ tal que $g_{s+1}(n) = g_s(n)$ si $1 \leq n \leq s$ y

$$g_{s+1}(s+1) = \begin{cases} g_s(s) + 2 & \text{si } g_s(s) \text{ es par y } \forall j \in \omega(g_s(s) \neq 2k_j) \\ 2j + 1 & \text{si } g_s(s) \text{ es par y } g_s(s) = 2k_j \text{ para algún } j \in \omega \\ 2k_j + 2 & \text{si } g_s(s) = 2j + 1 \text{ para algún } j \in \omega \end{cases}$$

Notemos que $g_s \subseteq g_{s+1}$ para cada $s \in \omega$ luego $g := \bigcup_{s \in \omega} g_s$ es una función con dominio ω y contradominio ω.

Afirmamos lo siguiente:

(iv) Para cualquier $n \in \omega$, $g(n) \leq^{\mathcal{M}} g(n + 1)$.

(v) Para cualquier $n \in \omega$, si $m < n$ entonces $g(m) \neq g(n)$.

(vi) Para cualquier $n \in \omega$, no existe $m \in \omega$ tal que $m \neq g(n)$, $m \neq g(n + 1)$ y $g(n) \leq^{\mathcal{M}} m \leq^{\mathcal{M}} g(n + 1)$.

El inciso *(iv)* es inmediato de la definición de g.

Demostremos el inciso *(v)* por inducción sobre n. Para $n = 0$ la afirmación es verdadera por vacuidad. Supongamos que el enunciado es cierto para n y veamos que se cumple para $n + 1$.

Si $g(n)$ es par y $g(n) \neq 2k_j$ para ningún $j \in \omega$ entonces $g(n + 1) = g(n) + 2$, luego $g(n + 1) \neq g(n)$.

Si $g(n)$ es par de la forma $2k_j$ para algún $j \in \omega$ entonces $g(n + 1) = 2j + 1$; si ocurriera que $g(n + 1) = g(n)$ entonces $g(n) = 2j + 1$, lo cual implicaría que un número par es igual a un número impar, lo cual es falso, por lo tanto $g(n+1) \neq g(n)$.

Si $g(n)$ es impar de la forma $2j+1$ entonces $g(n+1) = 2k_j + 2$. Nuevamente si $g(n+1) = g(n)$, tendríamos que un número impar es igual a un número par, lo cual es falso. Por lo tanto $g(n+1) \neq g(n)$.

En cualquier caso concluimos que $g(n+1) \neq g(n)$. Si ocurriera que $g(n+1) = g(m)$ para algún $m < n+1$ entonces por el inciso anterior y la hipótesis tenemos que $g(m) \leq^{\mathcal{M}} g(n)$ y $g(n) \leq^{\mathcal{M}} g(m)$. Por lo tanto, $g(m) = g(n)$, lo cual contradice nuestra hipótesis inductiva. Entonces debe ocurrir que $g(n+1) \neq g(m)$ para todo $m < n+1$. Esto demuestra el inciso *(v)*.

Demostremos el inciso *(vi)*. Por contradicción supongamos que existen $n, m \in \omega$ tales que $m \neq g(n)$, $m \neq g(n+1)$ y $g(n) \leq^{\mathcal{M}} m \leq^{\mathcal{M}} g(n+1)$.

Si $g(n) = 2s$, es decir es par y $g(n) \neq 2k_j$ para ningún $j \in \omega$ entonces $g(n+1) = g(n) + 2 = 2s + 2 = 2(s+1)$. Si $m = 2i$, es decir es par entonces como $g(n) \leq^{\mathcal{M}} m$ tenemos que $s \leq^{\mathcal{A}} i$ y como $m \leq^{\mathcal{M}} g(n+1)$ entonces $i \leq^{\mathcal{A}} s+1$. Si $i = s$ entonces $g(n) = m$, lo cual es falso. Si $i = s+1$ entonces $g(n+1) = m$, lo cual no ocurre. Por lo tanto m no es par. Si $m = 2i+1$, es decir es impar entonces como $g(n) \leq^{\mathcal{M}} m$ tenemos que $s \leq^{\mathcal{A}} k_i$ y como $m \leq^{\mathcal{M}} g(n+1)$ entonces $k_i + 1 \leq^{\mathcal{A}} s+1$. Por lo tanto $s = k_i$, lo cual implica que $g(n) = 2k_i$, la cual es una contradicción por la hipótesis sobre $g(n)$. Entonces m no es impar. Concluimos que este caso no sucede.

Si $g(n) = 2s$, es decir es par y $g(n) = 2k_j$ para algún $j \in \omega$ entonces $g(n+1) = 2j+1$. Si $m = 2i$, es decir es par entonces como $m \leq^{\mathcal{M}} g(n+1)$ entonces $i \leq^{\mathcal{A}} k_j$. Esto último implica que $2i \leq^{\mathcal{M}} 2k_j$, es decir que $m \leq^{\mathcal{M}} g(n)$. Como $g(n) \leq^{\mathcal{M}} m$ entonces $g(n) = m$ lo cual es falso. Por lo tanto m no es par. Si $m = 2i+1$, es decir es impar entonces como $g(n) \leq^{\mathcal{M}} m$ tenemos que $k_j \leq^{\mathcal{A}} k_i$ y como $m \leq^{\mathcal{M}} g(n+1)$ entonces $k_i \leq^{\mathcal{A}} k_j$, lo cual implica que $k_j = k_i$, es decir $i = j$. Luego $g(n+1) = m$, lo cual es falso. Entonces m no es impar. Por lo tanto este caso no sucede.

Sólo queda el caso en el que $g(n) = 2s+1$, es decir es impar. Entonces $g(n+1) = 2k_s+2$. Si $m = 2i$, es decir es par entonces como $g(n) \leq^{\mathcal{M}} m$ tenemos que $k_s+1 \leq^{\mathcal{A}} i$ y como $m \leq^{\mathcal{M}} g(n+1)$ entonces $i \leq^{\mathcal{A}} k_s+1$. Luego $i = k_s+1$. Entonces $2i = 2k_s+2$, es decir $m = g(n+1)$, lo cual es falso. Por lo tanto m no es par. Si $m = 2i+1$, es decir es impar entonces como $g(n) \leq^{\mathcal{M}} m$ tenemos que $k_s \leq^{\mathcal{A}} k_i$ y como $m \leq^{\mathcal{M}} g(n+1)$ entonces $k_i + 1 \leq^{\mathcal{A}} k_s + 1$. Por lo tanto $k_i = k_s$, lo cual implica que $i = s$. Entonces $g(n) = m$, lo cual es falso. Por consiguiente m no es impar. Por lo tanto este caso no sucede.

Como no sucede ninguno de los posibles casos entonces debe ser falso que existen $n, m \in \omega$ tales que $m \neq g(n)$, $m \neq g(n+1)$ y $g(n) \leq^{\mathcal{M}} m \leq^{\mathcal{M}} g(n+1)$. Lo cual demuestra el inciso *(vi)*.

Veamos que g es inyectiva. En efecto sean $n, m \in \omega$ tales que $g(n) = g(m)$.

Supongamos que $m \neq n$. Supongamos que $m < n$. Entonces por la observación *(v)* tenemos que $g(m) \neq g(n)$ lo cual es falso. Si $n < m$, de la misma forma obtenemos una contradicción. Por lo tanto debe ocurrir que $n = m$. Concluimos entonces que g es inyectiva.

Ahora veamos que g es sobreyectiva. Sea $m \in \omega$. Una observación anterior nos dice que m tiene una cantidad finita de $\leq^{\mathcal{M}}$-predecesores, digamos $l_m + 1$. Veamos que $g(l_m) = m$.

Supongamos por contradicción que $g(l_m) \neq m$. Primero notemos que si $g(l_m + 1) = m$ entonces, como para cada $n \leq l_m + 1$, $g(n) \leq^{\mathcal{M}} m$, la cantidad de $\leq^{\mathcal{M}}$-predecesores es mayor o igual a $l_m + 2$, lo cual es falso. Por lo tanto $g(l_m + 1) \neq m$. Ahora bien una observación anterior nos dice que $g(l_m) \leq^{\mathcal{M}} m$ ó $m \leq^{\mathcal{M}} g(l_m)$.

Si $g(l_m) \leq^{\mathcal{M}} m$ entonces como $g(l_m), g(l_m + 1) \neq m$ no puede ocurrir que $m \leq^{\mathcal{M}} g(l_m + 1)$.

Por lo tanto $g(l_m + 1) \leq^{\mathcal{M}} m$. Sin embargo para cada $n \leq l_m + 1$, $g(n) \leq^{\mathcal{M}} m$, la cantidad de $\leq^{\mathcal{M}}$-predecesores es mayor o igual a $l_m + 2$, lo cual es falso. Entonces no ocurre que $g(l_m) \leq^{\mathcal{M}} m$. Por lo tanto $m \leq^{\mathcal{M}} g(l_m)$.

Supongamos que $m \neq g(i)$ para ningún $i < l_m$. Entonces $m \leq^{\mathcal{M}} g(0)$ ó $g(0) \leq^{\mathcal{M}} m$. Si $m \leq^{\mathcal{M}} g(0)$ entonces como $0 \leq^{\mathcal{M}} m$, es decir $g(0) \leq^{\mathcal{M}} m$ tenemos que $g(0) = m$, lo cual contradice nuestra hipótesis. Por lo tanto $g(0) \leq^{\mathcal{M}} m$. Una observación anterior nos dice que $m \leq^{\mathcal{M}} g(1)$ ó $g(1) \leq^{\mathcal{M}} m$. La primera no puede ocurrir por la hipótesis y por la observación del inciso *(vi)*. Por lo tanto $g(1) \leq^{\mathcal{M}} m$. Sucesivamente seguimos este proceso hasta obtener que $g(l_m) \leq^{\mathcal{M}} m$. Luego $g(l_m) = m$, lo cual contradice nuestra hipótesis. Por lo tanto debe ocurrir que $m = g(i)$ para algún $i < l_m$.

Afirmamos que si $n \leq^{\mathcal{M}} m$ entonces $n = g(s)$ para algún $s \leq i$. Supongamos lo contrario, es decir que $n \neq g(s)$ para ningún $s \leq i$. Tenemos que $n \leq^{\mathcal{M}} g(0)$ ó $g(0) \leq^{\mathcal{M}} n$. Si $n \leq^{\mathcal{M}} g(0)$ entonces como se realizó en pasos anteriores, concluimos que $g(0) = n$, lo cual contradice nuestra hipótesis. Si $g(0) \leq^{\mathcal{M}} n$ entonces debe ocurrir que $g(1) \leq^{\mathcal{M}} n$. Siguiendo con este proceso tenemos que $g(i) \leq^{\mathcal{M}} n$, es decir $m \leq^{\mathcal{M}} n$. Por lo tanto $g(i) = m = n$, lo cual contradice nuestra hipótesis. Por consiguiente, si $n \leq^{\mathcal{M}} m$ entonces $n = g(s)$ para algún $s \leq i$.

Por otro lado observemos que si $s \leq i$ entonces $g(s)$ es un $\leq^{\mathcal{M}}$-predecesor de m. Por lo tanto, la cantidad de $\leq^{\mathcal{M}}$-predecesores de m es igual a $i + 1 \leq l_m < l_m + 1$, lo cual contradice la definición de $l_m + 1$. Por lo tanto no ocurre que $m \leq^{\mathcal{M}} g(l_m)$. Sin embargo esto contradice que debe ocurrir alguno de los casos $g(l_m) \leq^{\mathcal{M}} m$ ó $m \leq^{\mathcal{M}} g(l_m)$. Por lo tanto debe ser falso que $g(l_m) \neq m$. Entonces $g(l_m) = m$. Concluimos que g es sobreyectiva. Por lo tanto g es biyectiva.

Ahora veamos que dados $n, m \in \omega$, $n \leq^{\mathcal{A}} m$ si y sólo si $g(n) \leq^{\mathcal{M}} g(m)$. Que $n \leq^{\mathcal{A}} m$ implique que $g(n) \leq^{\mathcal{M}} g(m)$ se sigue de la observación *(iv)*. Ahora si $g(n) \leq^{\mathcal{M}} g(m)$ y $n = m$ entonces es claro que $n \leq^{\mathcal{M}} m$. Si $g(n) \leq^{\mathcal{M}} g(m)$ y $n \neq m$ supongamos que $m \leq^{\mathcal{M}} n$. Tenemos que $g(m) \leq^{\mathcal{M}} g(n)$. Luego $g(n) = g(m)$. Como g es inyectiva tenemos que $n = m$, lo cual contradice nuestra hipótesis. Por lo tanto $n \leq^{\mathcal{A}} m$. Concluimos que dados $n, m \in \omega$, $n \leq^{\mathcal{A}} m$ si y sólo si $g(n) \leq^{\mathcal{M}} g(m)$. Por lo tanto g es un isomorfismo.

Es sencillo ver que $\mathcal{A}$ es computable. Para ver que $\mathcal{M}$ es computable notemos que para verificar que un número es $\leq^{\mathcal{M}}$ que otro basta recurrir a $\leq^{A}$ y quizá a calcular de forma computable valores k_j para $j \in \omega$. Por lo tanto $\mathcal{M}$ es computable.

Veamos ahora que entre $\mathcal{A}$ y $\mathcal{M}$ no puede existir un isomorfismo computable. Supongamos que existe uno digamos h de $\mathcal{A}$ a $\mathcal{M}$. Definimos

$$S = \{(n, m) \in \omega \times \omega : \mathcal{A} \vDash \neg\exists x(n \neq x \wedge m \neq x \wedge n \leq^{\mathcal{A}} x \leq^{\mathcal{A}} m)\}$$

$$R = \{(n, m) \in \omega \times \omega : \mathcal{M} \vDash \neg\exists x(n \neq x \wedge m \neq x \wedge n \leq^{\mathcal{M}} x \leq^{\mathcal{M}} m)\}$$

Notemos que S es la relación de la función sucesor, por consiguiente es computable. Como consecuencia de que h es un isomorfismo entonces para cualesquiera $n, m \in \omega$, $\mathcal{A} \vDash \neg\exists x(n \neq x \wedge m \neq x \wedge n \leq^{\mathcal{A}} x \leq^{\mathcal{A}} m)$ si y sólo si $\mathcal{M} \vDash \neg\exists x(h(n) \neq x \wedge h(m) \neq x \wedge h(n) \leq^{\mathcal{M}} x \leq^{\mathcal{M}} h(m))$. Por lo tanto, $R(2n, 2n+2)$ si y sólo si $S(h(2n), h(2n+2))$. Sin embargo $R(2n, 2n+2)$ si y sólo si $n \neq k_j$ para ningún $j \in \omega$, lo cual ocurre si y sólo si $n \notin K$. En resumen, $n \notin K$ si y sólo si $S(h(2n), h(2n+2))$. Como h es computable podemos verificar de forma computable si $S(h(2n), h(2n+2))$, es decir podemos verificar de forma computable si $n \notin K$. Por lo tanto K es computable, lo cual es una contradicción. Concluimos que no puede existir un isomorfismo computable entre $\mathcal{A}$ y $\mathcal{M}$.

□

El ejemplo anterior nos deja claro que el hecho de que dos ω-presentaciones sean isomorfas y sean computables no significa que exista un isomorfismo computable entre ellas. En lo que sigue en esta sección mostraremos condiciones sobre una estructura computable para que podamos tener isomorfismos computables entre sus ω-presentaciones. También daremos condiciones para obtener el grado de Turing de un isomorfismo.

Definición 5.2.1. Dada una estructura $\mathcal{A}$, una **familia de Scott** de $\mathcal{A}$ es un conjunto $\mathcal{S}$ de fórmulas tal que para cada $\overline{a} \in A^{<\mathbb{N}}$ satisface una fórmula en $\mathcal{S}$ y si dos tuplas $\overline{a}, \overline{b} \in A^{<\mathbb{N}}$ satisfacen la misma fórmula entonces existe un automorfismo de $\mathcal{A}$ que manda a $\overline{a}$ en $\overline{b}$.

En resumen, una familia de Scott es un conjunto de fórmulas que caracterizan a las tuplas que son automórficas entre sí.

Definición 5.2.2. Sean $\mathcal{A}$ y $\mathcal{B}$ dos estructuras, decimos que $I \subseteq A^{<\mathbb{N}} \times B^{<\mathbb{N}}$ tiene la **propiedad de ida y vuelta** si y sólo si para cada $(\overline{a}, \overline{b}) \in I$ se tiene que:

(i) $D_{\mathcal{A}}(\overline{a}) = D_{\mathcal{B}}(\overline{b})$

(ii) Para cada $c \in A$, existe $d \in B$ tal que $(\overline{a}c, \overline{b}d) \in I$.

(iii) Para cada $d \in B$, existe $c \in A$ tal que $(\overline{a}c, \overline{b}d) \in I$.

El motivo de esta definición es generalizar el argumento de ida y vuelta utilizado para mostrar que dos órdenes lineales numerables, densos y sin puntos lineales deben ser isomorfos. Enunciamos a continuación algunos ejemplos de conjuntos de ida y vuelta.

Ejemplo 5.2.2. Sean $\mathcal{A}, \mathcal{B}$ dos estructuras isomorfas. Entonces,

(i) El conjunto $I := \{(\overline{a}, \overline{b}) \in A^{<\mathbb{N}} \times B^{<\mathbb{N}} : (\mathcal{A}, \overline{a}) \cong (\mathcal{B}, \overline{b})\}$ tiene la propiedad de ida y vuelta.

(ii) Si $\mathcal{S}$ es una familia de Scott para $\mathcal{A}$ entonces el conjunto

$$L_{\mathcal{A},\mathcal{B}} := \{(\overline{a}, \overline{b}) \in A^{<\mathbb{N}} \times B^{<\mathbb{N}} : (\exists \varphi \in \mathcal{S}) \quad \begin{matrix} (\mathcal{A} \vDash \varphi[x_i \to a_i : i < len(\overline{a})] \wedge \\ \wedge\ \mathcal{B} \vDash \varphi[x_i \to b_i : i < len(\overline{b})]) \end{matrix} \}$$

tiene la propiedad de ida y vuelta.

Demostración. *(i)* Si $(\overline{a}, \overline{b}) \in I$ entonces $(\mathcal{A}, \overline{a}) \cong (\mathcal{B}, \overline{b})$, es decir existe un isomorfismo f de $\mathcal{A}$ a $\mathcal{B}$ que manda $\overline{a}$ en $\overline{b}$ y que además preserva la satisfacción de las interpretaciones de las relaciones de $\mathcal{A}$ en $\mathcal{B}$. Por consiguiente, tanto $\overline{a}$ como $\overline{b}$ deben satisfacer las mismas fórmulas atómicas, por lo tanto $D_{\mathcal{A}}(\overline{a}) = D_{\mathcal{B}}(\overline{b})$. Ahora, si $c \in A$, entonces $f(c) \in B$ es tal que $(\mathcal{A}, \overline{a}c) \cong (\mathcal{B}, \overline{b}f(c))$. De manera similar verificamos la condición restante.

(ii) Sea $(\overline{a}, \overline{b}) \in L_{\mathcal{A},\mathcal{B}}$. Afirmamos que $(\overline{a}, \overline{b}) \in I$. Sea f un isomorfismo de $\mathcal{A}$ a $\mathcal{B}$. Dado que $\overline{a}$ y $\overline{b}$ satisfacen una fórmula de $\mathcal{S}$ en sus respectivas estructuras entonces $\overline{a}$ y $f^{-1}(\overline{b})$ también satisfacen dicha fórmula en $\mathcal{A}$. Por lo tanto, existe un automorfismo g de $\mathcal{A}$ que manda a $\overline{a}$ a $f^{-1}(\overline{b})$. La función $f \circ g$ es un isomorfismo de $(\mathcal{A}, \overline{a})$ a $(\mathcal{B}, \overline{b})$. Por lo tanto $(\overline{a}, \overline{b}) \in I$.

Por el inciso anterior, tenemos que $D_{\mathcal{A}}(\overline{a}) = D_{\mathcal{B}}(\overline{b})$. Dado $c \in A$, existe $d \in B$ tal que $(\overline{a}c, \overline{b}d) \in I$. Luego, como $\overline{a}c$ satisface alguna fórmula en $\mathcal{S}$ y como existe un isomorfismo de $\mathcal{A}$ a $\mathcal{B}$ que manda a $\overline{a}c$ en $\overline{b}d$ se debe cumplir que $\overline{b}d$ satisface la misma fórmula en $\mathcal{B}$ y por lo tanto $(\overline{a}c, \overline{b}d) \in L_{\mathcal{A},\mathcal{B}}$. La condición restante se verifica de manera similar.

□

Un resultado que nos será util más adelante es el siguiente:

Lema 5.2.1. *Sean $\mathcal{A}$ y $\mathcal{B}$ estructuras, $X \subseteq \omega$ e $I \subseteq A^{<\mathbb{N}} \times B^{\mathbb{N}}$ un conjunto X-c.e. con la propiedad de ida y vuelta, entonces para cada $(\overline{a}, \overline{b}) \in I$ existe un isomorfismo X-computable f de $\mathcal{A}$ a $\mathcal{B}$ que manda a $\overline{a}$ en $\overline{b}$.*

Demostración. Sea $(\overline{a}, \overline{b}) \in I$. Notemos que podemos buscar a dicha tupla de forma X-computable. Mostremos que existe un isomorfismo que manda $\overline{a}$ a $\overline{b}$. Definiremos a f como la unión de una sucesión de funciones parciales finitas.

Para ello primero notemos que $D_{\mathcal{A}}(\overline{a}) = D_{\mathcal{B}}(\overline{b})$ y por consiguiente $len(\overline{a}) = len(\overline{b})$. Ahora bien para cualesquiera $i, j < len(\overline{a})$, si φ es la fórmula $x_i = x_j$ entonces $\mathcal{A} \vDash \varphi[x_i \to a_i]$ si y sólo si $\mathcal{B} \vDash \varphi[x_j \to b_j]$, luego $a_i = a_j$ si y sólo si $b_i = b_j$. Definimos $f_0 : A \nrightarrow B$ tal que $f_0(a_i) = b_j$. Por la observación previa, tenemos que f_0 es en efecto una función parcial que además es inyectiva.

Supongamos ahora que hemos definido funciones parciales e inyectivas f_i tales que $f_i \subseteq f_{i+1}$ para cada $i \leq s$ y que formamos una tupla de la forma $(\overline{a}_s, \overline{b}_s) \in I$. Definimos $f_{s+1} : A \nrightarrow B$ tal que $f_{s+1} \upharpoonright \text{dom} f_s = f_s$ y tal que se le agregan los siguientes elementos. Para el menor elemento en A que no está en dom f_s, digamos c, tenemos que existe $d \in B$ tal que $(\overline{a}_s c, \overline{b}_s d) \in I$. Notemos que podemos buscar la tupla $(\overline{a}_s c, \overline{b}_s d)$ de forma X-computable. También para el menor elemento en B que no está en la imagen de f_s, digamos d', existe $c' \in A$ tal que $(\overline{a}_s cc', \overline{b}_s dd') \in I$. De la misma forma podemos buscar a dicha tupla de forma X-computable. Definimos $f_{s+1}(c) = d$, $f_{s+1}(c') = d'$, $\overline{a}_{s+1} = \overline{a}_s cc'$ y $b_{s+1} = \overline{b}_s dd'$.

Notemos que $len(\overline{a}_{s+1}) = len(\overline{b}_{s+1})$. Si $a = c$ y a está en la tupla $\overline{a}_s$ entonces c está en la tupla $\overline{a}_s$, lo cual implica que c está en dom f lo cual por construcción es falso. Por lo tanto a no está en la tupla $\overline{a}_s$. Por otro lado como $(\overline{a}_s c, \overline{b}_s d) \in I$ tenemos que $D_{\mathcal{A}}(\overline{a}_s c) = D_{\mathcal{B}}(\overline{b}_s d)$. Luego para cada $i < |\overline{a}_s|$ sean φ_i las fórmulas $x_{|\overline{a}_s|} = x_i$. Por lo anterior tenemos que $c = a$ para algún a en la tupla $\overline{a}_s$ si y sólo si $d = b$ para algún d en la tupla $\overline{b}_s$. Como lo primero es falso entonces debe ser falso que $d = b$ para algún d en la tupla $\overline{b}_s$.

De manera análoga podemos ver que c' es distinto de los elementos de la tupla $\overline{a}_s$ y de c y que d' es distinto de los elementos de la tupla $\overline{b}_s$ y de d. Por lo tanto f_{s+1} es una función parcial e inyectiva.

Definimos $f := \bigcup_{s \in \omega} f_s$. Como $f_s \subseteq f_{s+1}$ para cada $s \in \omega$ tenemos que f es una función. Notemos que dom $f = \omega$ ya que $s \in$ dom f_{s+1}. Como cada f_s es inyectiva tenemos que f es inyectiva. También que f es sobreyectiva ya que $s \in$ im g_s. Por lo tanto f es biyectiva.

Ahora bien, sea R_i una relación. Sea $\overline{q} \in A^{a_R(i)}$ tal que $\overline{q} \in R_i^{\mathcal{A}}$. Notemos que existe una tupla $\overline{a}_s$ tal que todos los elementos de $\overline{q}$ están en $\overline{a}_s$ y $len(\overline{a}_s) > i$. Como

$D_{\mathcal{A}}(\overline{a}_s) = D_{\mathcal{B}}(f(\overline{a}_s))$ entonces como $\overline{a}_s$ extiende a $\overline{q}$ entonces $f(\overline{q}) \in R_i^{\mathcal{B}}$. De manera similar podemos ver que si $f(\overline{q}) \in R_i^{\mathcal{B}}$ entonces $\overline{q} \in R_i^{\mathcal{A}}$. Por lo tanto concluimos que f es un isomorfismo.

Veamos que f es X-computable. En efecto, dado $n \in \omega$ entonces obtenemos los sucesivos valores de f a partir de la construcción de las funciones parciales f_s como se hizo anteriormente hasta encontrar el valor de n en una tupla $\overline{a}_s$ y devolvemos su valor correspondiente en la tupla $\overline{b}_s$. Este proceso lo podemos realizar de forma X-computable tal como se observó en la construcción.

□

Ahora combinemos nuestros resultados anteriores para mostrar una condición necesaria y suficiente para que una estructura computable tenga isomorfismos computables con sus ω-presentaciones computables y además que dichos isomorfismos se puedan obtener usando un único programa. Para ello damos las siguientes definiciones.

Definición 5.2.3. Sea $\mathcal{A}$ una estructura computable.

(i) Decimos que $\mathcal{A}$ es **uniforme y computablemente categórica** si y sólo si existe $e \in \omega$ tal que para cualquier ω-presentación computable $\mathcal{B}$ de $\mathcal{A}$ se cumple que $\Phi_e^{D(\mathcal{B})}$ es un isomorfismo de $\mathcal{B}$ a $\mathcal{A}$.

(ii) Diremos que $\mathcal{A}$ es **uniforme, relativa y computablemente categórica** si y sólo si existe $e \in \omega$ tal que para cualquier ω-presentación $\mathcal{B}$ de $\mathcal{A}$ se tiene que $\Phi_e^{D(\mathcal{B})}$ es un isomorfismo de $\mathcal{B}$ a $\mathcal{A}$.

A continuación damos el resultado mencionado con anterioridad.

Teorema 5.2.1. *Sea $\mathcal{A}$ una estructura computable. Las siguientes son equivalentes:*

(i) Existe una familia de Scott computable enumerable para $\mathcal{A}$ que consiste de fórmulas existenciales.

(ii) $\mathcal{A}$ es uniforme, relativa y computablemente categórica.

(iii) $\mathcal{A}$ es uniforme y computablemente categórica.

Demostración. Veamos que *(i)* implica *(ii)*. Sean $\mathcal{S}$ una familia de Scott para $\mathcal{A}$ como en la hipótesis, $\mathcal{B}$ una ω-presentación de $\mathcal{A}$ y g un isomorfismo que va de $\mathcal{A}$ a $\mathcal{B}$. Afirmamos que $L_{\mathcal{A},\mathcal{B}}$ es c.e. en $D(\mathcal{B})$. En efecto, primero obtenemos una tupla de la forma $(\overline{a}, \overline{b})$ y obtenemos de forma computable una fórmula en $\mathcal{S}$. Con $D(\mathcal{A})$ verificamos si dicha fórmula es verdadera en $\mathcal{A}$ tras sustituir sus variables por los elementos de $\overline{a}$. Al mismo tiempo con $D(\mathcal{B})$ verificamos si esa fórmula es verdadera en $\mathcal{B}$ tras sustituir sus variables por los elementos de $\overline{b}$. Mientras se realiza esta verificación, conseguimos la siguiente fórmula en $\mathcal{S}$ y realizamos la misma verificación. Continuamos con este proceso hasta encontrar una fórmula

en $\mathcal{S}$ que requerimos. Cuando esto ocurra, el programa devuelve la tupla $(\overline{a}, \overline{b})$ y termina. Observemos que este programa puede correr indefinidamente si no encuentra dicha fórmula. Como $\mathcal{A}$ es computable, este programa hace a $L_{\mathcal{A},\mathcal{B}}$ un conjunto $D(\mathcal{B})$-computable enumerable.

Notemos que $L_{\mathcal{A},\mathcal{B}} \neq \emptyset$ ya que $(0, g(0)) \in L_{\mathcal{A},\mathcal{B}}$. Definimos un programa Φ_e tal que con $D(\mathcal{B})$ obtenga el primer elemento enumerado en $L_{\mathcal{A},\mathcal{B}}$. Luego para la tupla obtenida, con $D(\mathcal{B})$, podemos construir un isomorfismo que va de $\mathcal{A}$ a $\mathcal{B}$, es decir podemos obtener los valores de dicho isomorfismo. Con dicha obtención de valores también podemos obtener los que corresponden a su función inversa, la cual es un isomorfismo que va de $\mathcal{B}$ a $\mathcal{A}$. Finalmente el programa devuelve los valores de dicha función inversa. Luego $\Phi_e^{D(\mathcal{B})}$ es un isomorfismo que va de $\mathcal{B}$ a $\mathcal{A}$.

Que *(ii)* implica *(iii)* se sigue de las definiciones.

Veamos que *(iii)* implica *(i)*. Sea Φ_e tal que si $\mathcal{B}$ es una presentación de $\mathcal{A}$ entonces $\Phi_e^{D(\mathcal{B})}$ es un isomorfismo que va de $\mathcal{B}$ a $\mathcal{A}$.

Afirmamos que para cada $\overline{q} \in A^{<\omega}$ y $k \in \omega$ con $k \leq len(\overline{q})$, si $\Phi_e^{D_{\mathcal{A}}(\overline{q})}$ converge para cada $0 \leq n \leq k-1$ entonces existe un automorfismo de $\mathcal{A}$ que manda $\overline{q} \upharpoonright k$ a $\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright k$.

Sea $q = (q_0, ..., q_{t-1}) \in A^{<\omega}$ tal que $\Phi_e^{D_{\mathcal{A}}(\overline{q})}$ converge para cada $0 \leq n \leq k-1$.

Definimos $g : \omega \to A$ tal que $g(i) = q_i$ para cada $0 \leq i \leq t-1$ y $g(i+1) = \min_{j \in \omega}\{\forall l \leq i (j \neq g(l))\}$ para cada $i \geq t-1$. Por la definición de g tenemos que g es una función computable y sobreyectiva. Sea $\mathcal{B} := g^{-1}(\mathcal{A})$ el pull-back de $\mathcal{A}$ respecto de g. Notemos que $\mathcal{B}$ es una ω-presentación de $\mathcal{A}$ y que es computable. Por lo tanto $\Phi_e^{D(\mathcal{B})}$ es un isomorfismo que va de $\mathcal{B}$ a $\mathcal{A}$.

Observemos que g^{-1} es un isomorfismo que va de $\mathcal{A}$ a $\mathcal{B}$ que manda a $g \upharpoonright k$ a $(0, 1, ..., k-1)$. Por lo tanto $\Phi_e^{D(\mathcal{B})}$ es un isomorfismo que manda a $(0, 1, ..., k-1)$ a $\Phi_e^{D(\mathcal{B})} \upharpoonright k$. Por lo tanto $\Phi_e^{D(\mathcal{B})} \circ g^{-1}$ es un automorfismo de $\mathcal{A}$ que manda $g \upharpoonright k$ a $\Phi_e^{D(\mathcal{B})} \upharpoonright k$. Como $D_{\mathcal{A}}(\overline{q}) \subseteq D(\mathcal{B})$ y $\Phi_e^{D_{\mathcal{A}}(\overline{q})}$ converge para cada $0 \leq n \leq k-1$ entonces $\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright k = \Phi_e^{D(\mathcal{B})} \upharpoonright k$. Por otro lado como g es una extensión de la tupla $\overline{q}$ y $k < len(\overline{q})$ entonces $g \upharpoonright k = \overline{q} \upharpoonright k$. De todo lo anterior concluimos que existe un automorfismo de $\mathcal{A}$ que manda $\overline{q} \upharpoonright k$ a $\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright k$, a saber $\Phi_e^{D(\mathcal{B})} \circ g^{-1}$.

Prosigamos a definir una familia de Scott c.e. de fórmulas existenciales para $\mathcal{A}$. Sea $\overline{a} \in A^{<\omega}$ y definimos $k = len(\overline{a})$. Como $\Phi_e^{D(\mathcal{B})}$ converge para cada $0, 1, ..., k-1$ y $D(\mathcal{B}) = \bigcup_{s \in \omega} D_{\mathcal{A}}(g \upharpoonright s)$ entonces existe $s_0 \in \omega$ tenemos que $\Phi_e^{D_{\mathcal{A}}(g \upharpoonright s_0)}$ converge en $0, 1, ..., k-1$. Notemos que podemos extender a $\overline{a}$ a una tupla $\overline{q}$ tal que $\Phi_e^{D_{\mathcal{A}}(\overline{q})}$ converge en $0, 1, ..., k-1$.

En efecto denominamos a $\overline{a}$ como $\overline{a}_0$. Supongamos que hemos obtenido una tupla $\overline{a}_t$. Verificamos si $\Phi_e^{D_{\mathcal{A}}(\overline{a}_t)}$ converge en $0, 1, ..., k-1$. Sin esperar la respuesta de si converge agregamos un nuevo elemento distinto a los elementos de $\overline{a}_t$ y denominamos a la tupla obtenida como $\overline{a}_{t+1}$. Repetimos el ciclo con la tupla anterior hasta que algún $\Phi_e^{D_{\mathcal{A}}(\overline{a}_t)}$ se haya detenido en $0, 1, ..., k-1$. Esto debe suceder ya que en algún momento tendremos los elementos de $g \upharpoonright s_0$ en una tupla $\overline{a}_{t_0}$. Por lo tanto al menos $\Phi_e^{D_{\mathcal{A}}(\overline{a}_{t_0})}$ deberá converger en $0, 1, ..., k-1$. Definimos como $\overline{q}$ a la primer tupla que encontremos que cumple que $\Phi_e^{D_{\mathcal{A}}(\overline{q})}$ converge en $0, 1, ..., k-1$. Por lo mencionado anteriormente, podemos observar que podemos obtener a $\overline{q}$ de forma computable. Por la observación anterior debe existir un automorfismo de $\mathcal{A}$ que manda $\overline{a} = \overline{q} \upharpoonright k$ a $\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright k$.

Definimos la fórmula existencial

$$\varphi_{\overline{a}}(\overline{x}) = \exists \overline{y}(\overline{y} \supseteq \overline{x} \wedge D_{\mathcal{A}}(\overline{y}) = D_{\mathcal{A}}(\overline{q}))$$

No es claro que $\varphi_{\overline{a}}(\overline{x})$ es una fórmula existencial en el lenguaje de $\mathcal{A}$ para ello observemos que $\varphi_{\overline{a}}$ ese puede escribir como

$$\exists y_0, ..., y_{k-1}, y_k, ..., y_{len(\overline{q})-1}((y_0 = x_0 \wedge \cdots \wedge y_{k-1} = x_{k-1}) \wedge \varphi^{lc}_{t(D_{\mathcal{A}}(\overline{q}))}(y_0, ..., y_{len(\overline{q}-1)}))$$

donde $\overline{y}$ abrevia a $(y_0, ..., y_{len(\overline{q})-1})$ y $\overline{x}$ a $(x_0, ..., x_{k-1})$.

Notemos la anterior fórmula es en efecto una fórmula existencial en el lenguaje de $\mathcal{A}$. Observemos que $=$ es un símbolo que tiene cualquier estructura que se interpreta como la igualdad usual de los números naturales. La fórmula $\varphi^{lc}_{t(D_{\mathcal{A}}(\overline{q}))}(y_0, ..., y_{len(\overline{q}-1)}))$ es la fórmula de la observación 4,2,1. Además como $\mathcal{A}$ es computable podemos obtener los valores de $D_{\mathcal{A}}(\overline{q})$ de forma computable. Con dichos valores y con una enumeración de las primeras l_k fórmulas atomicas en las variables $x_0, ..., x_{len(\overline{q})-1}$ y en los símbolos relacionales $R_0, ..., R_{len(\overline{q})-1}$ podemos construir de forma computable la fórmula $\varphi^{lc}_{t(D_{\mathcal{A}}(\overline{q}))}$. Por lo tanto dada una tupla finita no vacía $\overline{a}$ podemos construir de forma computable a la fórmula $\varphi_{\overline{a}}$. Por consiguiente el conjunto $\{\varphi_{\overline{a}} : \overline{a} \in M^{<\mathbb{N}}\}$ es computable enumerable.

Observemos que $\mathcal{A} \vDash \exists y_0, ..., y_{k-1}, y_k, ..., y_{len(\overline{q})-1}(\varphi^{lc}_{t(D_{\mathcal{A}}(\overline{q}))}(y_0, ..., y_{len(\overline{q}-1)}))$ equivale a que existen $c_0, ..., c_{len(\overline{q})-1}$ tal que $D_{\mathcal{A}}(\overline{q}) \subseteq D_{\mathcal{A}}(c_0, ..., c_{len(\overline{q})-1})$. Como $\overline{q}$ y $(c_0, ..., c_{len(\overline{q})-1})$ tienen la misma longitud, lo anterior es equivalente a que existen $c_0, ..., c_{len(\overline{q})-1}$ tal que $D_{\mathcal{A}}(\overline{q}) = D_{\mathcal{A}}(c_0, ..., c_{len(\overline{q})-1})$.

Entonces dada una tupla $\overline{b} = (b_0, ..., b_{k-1}) \in M^{<\omega}$, tenemos que $\mathcal{A} \vDash \varphi_{\overline{a}}[x_j \to b_j : j < k]$ si y sólo si existen $c_0, ..., c_{len(\overline{q})-1}$ tal que $\overline{c} \supseteq \overline{b}$ y $D_{\mathcal{A}}(\overline{c}) = D_{\mathcal{A}}(\overline{q})$.

La tupla $\overline{a}$ satisface $\varphi_{\overline{a}}$ ya que la tupla que la extiende es $\overline{q}$ y por lo tanto se sigue la igualdad de los diagramas. Por otro lado si una tupla $\overline{b}$ satisface $\varphi_{\overline{a}}$ entonces existe una tupla $\overline{c}$ que extiende a $\overline{b}$ y $D_{\mathcal{A}}(\overline{c}) = D_{\mathcal{A}}(\overline{q})$. Como $\Phi_e^{D_{\mathcal{A}}(\overline{q})}$ converge en $0, 1, ..., k-1$ entonces $\Phi_e^{D_{\mathcal{A}}(\overline{c})}$ también converge en $0, 1, ..., k-1$, más aún $\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright k = \Phi_e^{D_{\mathcal{A}}(\overline{c})} \upharpoonright k$.

Por una observación anterior debe existir un automorfismo de $\mathcal{A}$ que manda $\overline{a} = \overline{q} \upharpoonright k$ a $\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright k = \Phi_e^{D_{\mathcal{A}}(\overline{c})} \upharpoonright k$. Por otro lado debe existir un automorfismo de $\mathcal{A}$ que manda $\Phi_e^{D_{\mathcal{A}}(\overline{c})} \upharpoonright k$ a $\overline{c} \upharpoonright k = \overline{b}$. Por lo tanto debe existir un automorfismo de $\mathcal{A}$ que manda $\overline{a}$ a $\overline{b}$. Lo anterior nos hace concluir que el conjunto $\{\varphi_{\overline{a}} : \overline{a} \in M^{<\mathbb{N}}\}$ es una familia de Scott computable enumerable de fórmulas existenciales.

□

5.3. Computabilidad de isomorfismos entre ω-presentaciones de una estructura (versión no uniforme)

Hemos dado una caracterización de cuando podemos obtener de manera uniforme isomorfismos de una estructura computable a sus ω-presentaciones. ¿Qué sucede con la versión no uniforme? A continuación daremos la versión no uniforme para que existan isomorfismos computables entre ω-presentaciones de una estructura.

Daremos las definiciones correspondientes para que podamos demostrar una caracterización similar a la anterior.

Definición 5.3.1. Sea $\mathcal{A}$ una estructura. Decimos que $R \subseteq \mathbb{N} \times A^{<\mathbb{N}}$ es **relativo e intrínsecamente computable enumerable (r.i.c.e.)** respecto a $\mathcal{A}$ si y sólo si para toda ω-presentación $(\mathcal{B}, R)$ de $(\mathcal{A}, R)$ se tiene que $R^{\mathcal{B}}$ es $D(\mathcal{B})$-computable enumerable.

La definición anterior nos dice que un conjunto R cuyos elementos definen nuevas relaciones en una estructura $\mathcal{A}$ es r.i.c.e. si y sólo si su interpretación en una ω-presentación $\mathcal{B}$ de $\mathcal{A}$ se puede enumerar sólo con la parte estructural de $\mathcal{B}$.

Podemos dar una definición más de la noción anterior. Dicha definición se define en términos sintácticos e introduce una nueva clase de fórmulas a las ya utilizadas hasta el momento.

Definición 5.3.2. Sea $\mathcal{L}$ un lenguaje predicativo.

(i) Una **$\mathcal{L}$-fórmula infinita** Σ_1 (Π_1 respectivamente) es una disyunción (conjunción) a lo más numerable de fórmulas existenciales (universales) sobre un conjunto finito de variables, las cuales representamos de la siguiente forma:

Si $\{\varphi_i^{\exists} : i \in \omega\}$ es una enumeración de las fórmulas existenciales sobre un conjunto finito de variables entonces toda fórmula infinita Σ_1 es de la forma

$$\bigvee_{i\in I} \varphi_i^{\exists}(\overline{x})$$

O bien si $\{\varphi_i^{\forall} : i \in \omega\}$ es una enumeración de las fórmulas universales sobre un conjunto finito de variables entonces toda fórmula infinita Π_1 es de la forma

$$\bigwedge_{i\in I} \varphi_i^{\forall}(\overline{x})$$

donde en ambos casos $\emptyset \neq I \subseteq \omega$

(ii) Una **$\mathcal{L}$-fórmula infinita** Σ_2 (Π_2 respectivamente) es una disyunción (conjunción) a lo más numerable de fórmulas existenciales (universales) sobre un conjunto finito de variables de fórmulas Π_1 (Σ_1 respectivamente), las cuales representaremos de la siguiente forma:

Si $\{\varphi_i^{\forall} : i \in \omega\}$ es una enumeración de las fórmulas universales sobre un conjunto finito de variables entonces toda fórmula infinita Σ_2 es de la forma

$$\bigvee_{j\in J} \exists \overline{y}_j \bigwedge_{i\in I} \varphi_i^{\forall}(\overline{x}, \overline{y_j})$$

O bien si $\{\varphi_i^{\exists} : i \in \omega\}$ es una enumeración de las fórmulas existenciales sobre un conjunto finito de variables entonces toda fórmula infinita Π_2 es de la forma

$$\bigwedge_{j\in J} \forall \overline{y}_j \bigvee_{i\in I} \varphi_i^{\exists}(\overline{x}, \overline{y}_j)$$

donde en ambos casos $\emptyset \neq I, J \subseteq \omega$

Es de nuestro interés interpretar una fórmula infinita en una estructura y definir una noción de verdad sobre ella. Para ello introducimos una noción de satisfactiblidad para este tipo de fórmulas tal como lo hicimos con las fórmulas finitas.

Definición 5.3.3. Sean $\mathcal{L}$ un lenguaje predicativo y $\mathcal{A}$ una $\mathcal{L}$-estructura.

(i) *(a)* Sean $\varphi(\overline{x})$ una fórmula infinita Σ_1 de la forma

$$\bigvee_{i\in I} \varphi_i^{\exists}(\overline{x})$$

y $\overline{a} = (a_0, ..., a_{len(\overline{x})-1}) \in A^{len(\overline{x})}$. Diremos que $\mathcal{A}$ satisface φ en $\overline{a}$ si y sólo si existe $i \in I$ tal que $\mathcal{A} \vDash \varphi_i^{\exists}[x_l \to a_l : l < len(\overline{x})]$.

(b) Sean $\varphi(\overline{x})$ una fórmula infinita Π_1 de la forma

$$\bigwedge_{i\in I} \varphi_i^{\forall}(\overline{x})$$

y $\overline{a} = (a_0, ..., a_{len(\overline{x})-1}) \in A^{len(\overline{x})}$. Diremos que $\mathcal{A}$ satisface φ en $\overline{a}$ si y sólo si para todo $i \in I$ tal que $\mathcal{A} \vDash \varphi_i^{\forall}[x_l \to a_l : l < len(\overline{x})]$.

(ii) *(a)* Sean $\varphi(\overline{x})$ una fórmula infinita Σ_2 de la forma

$$\bigvee_{j\in J} \exists \overline{y}_j \bigwedge_{i\in I} \varphi_i^{\forall}(\overline{x}, \overline{y_j})$$

y $\overline{a} = (a_0, ..., a_{len(x)-1}) \in A^{len(x)}$. Diremos que $\mathcal{A}$ satisface φ en $\overline{a}$ si y sólo si existen $j \in J$ y $\overline{b}_j = (b_0, ..., b_{len(\overline{y}_j)-1}) \in A^{len(\overline{y}_j)}$ tal que para todo $i \in I$ se tiene que $\mathcal{A} \vDash \varphi_i^{\forall}[x_l \to a_l, y_j \to b_j : l < len(x), j < len(\overline{y}_j)]$ donde $\overline{y}_j = (y_0, ..., y_{len(\overline{y}_j)-1})$.

(b) Sean $\varphi(\overline{x})$ una fórmula infinita Π_2 de la forma

$$\bigwedge_{j\in J} \forall \overline{y}_j \bigvee_{i\in I} \varphi_i^{\exists}(\overline{x}, \overline{y_j})$$

y $\overline{a} = (a_0, ..., a_{len(\overline{x})-1}) \in A^{len(\overline{x})-1}$. Diremos que $\mathcal{A}$ satisface φ en $\overline{a}$ si y sólo si para todo $j \in J$ y todo $\overline{b}_j = (b_0, ..., b_{len(\overline{y}_j)-1}) \in A^{len(\overline{y}_j)}$ existe $i \in I$ tal que $\mathcal{A} \vDash \varphi_i^{\exists}[x_l \to a_l, y_j \to b_j : l < len(x), j < len(\overline{y}_j)]$ donde $\overline{y}_j = (y_0, ..., y_{len(\overline{y}_j)-1})$.

En cualquier caso abreviaremos que φ ó $\varphi(\overline{x})$ satisface a $\overline{a}$ en $\mathcal{A}$ por $\mathcal{A} \vDash \varphi[x_l \to \overline{a} : l < len(\overline{a})]$.

Como notación, denotamos por $\langle \overline{x} \rangle$ a la tupla finita de códigos de la tupla de variables $\overline{x}$.

Ahora bien, queremos hacer una distinción de entre las fórmulas infinitas que hemos dado. La distinción introduce la noción de computabilidad entre fórmulas infinitas.

Definición 5.3.4. Sea $\mathcal{L}$ un lenguaje predicativo.

(i) Sea $\varphi(\overline{x})$ una $\mathcal{L}$-fórmula infinita Σ_1 (Π_1 respectivamente). Diremos que $\varphi(\overline{x})$ es **computable**, lo cual nos referiremos al decir $\varphi(\overline{x})$ es Σ_1^c (Π_1^c respectivamente) si y sólo si el conjunto de índices de las fórmulas existenciales (universales) que componen la disyunción (conjunción) que hace a $\varphi(\overline{x})$ una fórmula Σ_1 (Π_1

respectivamente) es computable enumerable. Si W_e es dicho conjunto entonces asociamos la tupla $(0, 1, \langle\overline{x}\rangle, e)$ ($(1, 1, \langle\overline{x}\rangle, e)$ respectivamente) a dicha fórmula, donde $\overline{x}$ son las variables libres de $\varphi(\overline{x})$. Diremos que dicha tupla es el código de $\varphi(\overline{x})$.

(ii) Sea $\varphi(\overline{x})$ una $\mathcal{L}$-fórmula infinita Σ_2 (Π_2 respectivamente). Diremos que $\varphi(\overline{x})$ es **computable**, lo cual nos referiremos al decir $\varphi(\overline{x})$ es Σ_2^c (Π_2^c respectivamente) si y sólo si

(a) Toda fórmula Π_1 (Σ_1) que forma parte de la disyunción (conjunción) que hace a $\varphi(\overline{x})$ una fórmula Σ_2 (Π_2 respectivamente) es computable. Esto es que para toda fórmula Π_1 (Σ_1), el conjunto de índices de las fórmulas universales (existenciales) que componen la conjunción (disyunción) que hace a dicha fórmula Π_1 (Σ_1 respectivamente) es computable enumerable.

(b) El conjunto de las tuplas $(i, \overline{y})$ donde i es el número asociado a la tupla $(1, 1, \langle\overline{x}\rangle, e')$ ($(0, 1, \langle\overline{x}\rangle, e')$ respectivamente) que corresponde a una fórmula ψ que es Π_1 (Σ_1 respectivamente) que tiene como variables libres a $\overline{x}$ y que forma parte de la disyunción (conjunción) que hace a $\varphi(\overline{x})$ una fórmula Σ_2 (Π_2 respectivamente) ocurriendo de la forma $\exists\overline{y}\ \psi$ (o bien $\forall\overline{y}\ \psi$), es computable enumerable. Si W_e es dicho conjunto entonces asociamos la tupla $(0, 2, \langle\overline{x}\rangle, e)$ ($(1, 2, \langle\overline{x}\rangle, e)$ respectivamente) a dicha fórmula. Diremos que dicha tupla es el código de $\varphi(\overline{x})$.

Una observación que se sigue de las definiciones anteriores es la siguiente:

Proposición 5.3.1. *Sea $\mathcal{L}$ un lenguaje predicativo.*

(i) El conjunto de códigos de $\mathcal{L}$-fórmulas infinitas Σ_1^c (Π_1^c respectivamente) es numerable. Más aún, es computable enumerable.

(ii) El conjunto de códigos de $\mathcal{L}$-fórmulas infinitas Σ_2^c (Π_2^c respectivamente) es numerable. Más aún, es computable enumerable.

Demostración. *(i)* Primero demostremos que para cada $j \in \omega$ con $j \geq 1$, el conjunto de los códigos de las fórmulas infinitas Σ_1^c que tienen exactamente j variables libres es computable enumerable. A partir de $n \in \omega$ enumeramos una tupla de la forma $(0, 1, \langle\overline{x}\rangle, e)$ donde $len(\overline{x}) = j$ y $\boldsymbol{\varphi_{j+1}}(\langle\overline{x}\rangle, e) = n$, donde $\boldsymbol{\varphi_{j+1}}$ es la biyección computable de las tuplas de números naturales de longitud $j + 1$ a ω vista con anterioridad. Esta tupla corresponderá a la fórmula Σ_1^c de la forma:

$$\bigvee_{i \in W_e} \varphi_i^{\exists}(\overline{x})$$

Sea $f_j : \omega \to \omega^{j+3}$ tal que $f_j(n) = (0, 1, \langle\overline{x}\rangle, e)$. Notemos que f_j es inyectiva y computable. Observemos que toda fórmula Σ_1^c con j-variables libres tiene

asignado un código de la forma $(0, 1, \langle\overline{x}\rangle, e)$ donde $\overline{x}$ son sus variables libres y W_e es el conjunto de índices de las fórmulas existenciales que componen la disyunción que hace a dicha fórmula Σ_1^c. Sea $n = \boldsymbol{\varphi_{j+1}}(\langle\overline{x}\rangle, e)$. Se sigue que la tupla asignada a n es $(0, 1, \langle\overline{x}\rangle, e)$. Por lo tanto im f_j es el conjunto de códigos de fórmulas Σ_1^c con j variables libres. Concluimos que el conjunto de fórmulas infinitas Σ_1^c con j-variables libres es computable enumerable.

Dicho esto, mostremos que el conjunto de códigos de las fórmulas computables Σ_1^c es computable enumerable. Sea $f : \omega \to \omega^{<\mathbb{N}}$ la función que dado $n \in \omega$, obtiene una tupla $(k, m) \in \omega^2$ con la función biyectiva y computable $\boldsymbol{\varphi_2}^{-1}$ vista con anterioridad y finalmente devuelve $f_k(m)$, cuyo valor depende de φ_{k+1}, la cual por construcción sólo depende de $\boldsymbol{\varphi_2}$. Por lo tanto f es computable. Sean $n_1, n_2 \in \omega$ tales que $f(n_1) = f(n_2)$. Supongamos que $\boldsymbol{\varphi_2}(k_i, m_i) = n_i$ para $i = 1, 2$. Luego $f_{k_1}(m_1) = f_{k_2}(m_2)$. Al tener ambas tuplas la misma longitud, concluimos que $k_1 = k_2$. Como $f_{k_1}(m_1) = f_{k_1}(m_2)$ entonces $m_1 = m_2$. Por lo tanto podemos concluir que $n_1 = n_2$. Concluimos que f es inyectiva. Veamos que im f_j es el conjunto de códigos de fórmulas Σ_1^c. Para ello sea $(0, 1, \langle\overline{x}\rangle, e)$ el código de una fórmula computable Σ_1^c. Supongamos que $len(\overline{x}) = j$ entonces existe $m \in \omega$ tal que $f_j(m) = (0, 1, \langle\overline{x}\rangle, e)$. Sea $n = \boldsymbol{\varphi_2}(j, m)$ entonces $f(n) = f_j(m) = (0, 1, \langle\overline{x}\rangle, e)$. Por lo tanto im f es el conjunto de códigos de fórmulas Σ_1^c. Concluimos que dicho conjunto es computable enumerable.

La demostración para fórmulas Π_1^c es similar.

(ii) La demostración para fórmulas Σ_2^c y Π_2^c es similar a la demostración para fórmulas Σ_1^c.

□

Notación. *(i)* Denotamos por $\{\varphi_i^{j,\Sigma_1^c} : i \in \omega\}$ al conjunto de códigos de fórmulas Σ_1^c con j-variables libres. Y denotamos por $\{\varphi_i^{\Sigma_1^c} : i \in \omega\}$ al conjunto de códigos de fórmulas Σ_1^c.

(ii) Denotamos por $\{\varphi_i^{j,\Pi_1^c} : i \in \omega\}$ al conjunto de códigos de fórmulas Π_1^c con j variables libres. Y denotamos por $\{\varphi_i^{\Pi_1^c} : i \in \omega\}$ al conjunto de códigos de fórmulas Π_1^c.

(iii) Denotamos por $\{\varphi_i^{j,\Sigma_2^c} : i \in \omega\}$ al conjunto de códigos de fórmulas Σ_2^c con j variables libres. Y denotamos por $\{\varphi_i^{\Sigma_2^c} : i \in \omega\}$ al conjunto de códigos de fórmulas Σ_2^c.

(iv) Denotamos por $\{\varphi_i^{j,\Pi_2^c} : i \in \omega\}$ al conjunto de códigos de fórmulas Π_2^c con j variables libres. Y denotamos por $\{\varphi_i^{\Pi_2^c} : i \in \omega\}$ al conjunto de códigos de fórmulas Π_2^c.

A partir de ahora, dada una fórmula φ que es Σ_i^c con $i = 1, 2$ denotamos por $\ulcorner\varphi\urcorner$ a su tupla asociada como en la proposición anterior.

A continuación enunciaremos un resultado que nos ayudará a caracterizar las satisfacción de una fórmula por una tupla en una estructura.

Proposición 5.3.2. *Existe $e \in \omega$ tal que para toda estructura $\mathcal{A}$ y toda ω-presentación $\mathcal{M}$ de una estructura $\mathcal{A}$, si φ es Σ_1^c entonces para todo $\overline{a} \in M^{<\mathbb{N}}$, $\boldsymbol{\psi}(\ulcorner\varphi\urcorner, \overline{a}) \in W_e^{D(\mathcal{M})}$ si y sólo si $\mathcal{M} \vDash \varphi[x_l \to a_l : l < len(\overline{x})]$.*

Demostración. Sea $\Phi_e^{D(\mathcal{M})}$ la función inducida por el siguiente programa que utiliza a $D(\mathcal{M})$ como oráculo. Dado $n \in \omega$, obtenemos $\boldsymbol{\psi}^{-1}(n)$. Con lo cual obtenemos una tupla finita de la forma $(n, m_1, ..., m_k)$. Verificamos si $\boldsymbol{\psi}^{-1}(n)$ tiene la forma de una tupla $(0, 1, \langle\overline{x}\rangle, e')$. En caso de que esto no sea así el programa corre indefinidamente. En caso contrario obtenemos el primer elemento enumerado por $\Phi_{e'}$ y lo tomamos como el índice de una fórmula existencial. Descomponemos dicha fórmula y con $D(\mathcal{M})$ verificamos si dicha fórmula se satisface por $m_1, ..., m_k$ en $\mathcal{M}$. Este proceso puede no detenerse ya que al tratarse de una fórmula existencial en algunos casos tendremos que buscar tuplas finitas que satisfagan dicha fórmula. Mientras esto ocurre repetimos el proceso con las demás fórmulas existenciales que aparezcan. Detenemos este proceso en cuanto encontremos una fórmula existencial que es satisfecha por $m_1, ..., m_k$ en $\mathcal{M}$. Ahora verifiquemos que e es el número natural que satisface lo requerido por el resultado.

Sea φ una fórmula Σ_1^c y $\overline{m} = (m_1, ..., m_k) \in M^{<\mathbb{N}}$. Si $\Phi_e^{D(\mathcal{M})}$ está definida en $\boldsymbol{\psi}(\ulcorner\varphi\urcorner, \overline{m})$ entonces durante la ejecución del programa anterior se debió encontrar una fórmula existencial que es satisfecha por $m_1, ..., m_k$ en $\mathcal{M}$, lo cual implica que φ es satisfecha por $m_1, ..., m_k$ en $\mathcal{M}$. Por otro lado si φ es satisfecha por $m_1, ..., m_k$ en $\mathcal{M}$ entonces en algún momento el programa anterior encontrará una fórmula existencial que es satisfecha por $m_1, ..., m_k$ en $\mathcal{M}$. Lo cual implica quedicho programa se detendrá. Por lo tanto $\boldsymbol{\psi}(\ulcorner\varphi\urcorner, m_1, ..., m_k) \in W_e^{D(\mathcal{M})}$.

Con lo cual concluimos que si φ es Σ_1^c entonces para todo $\overline{a} \in M^{<\mathbb{N}}$, $\boldsymbol{\psi}(\ulcorner\varphi\urcorner, \overline{a}) \in W_e^{D(\mathcal{M})}$ si y sólo si $\mathcal{M} \vDash \varphi[x_l \to a_l : l < len(\overline{x})]$.

□

La proposición anterior nos servirá para develar la relación que existe entre fórmulas Σ_1^c y conjuntos r.i.c.e. respecto a una estructura. Para ello primero diremos cuando un subconjunto de $\mathbb{N} \times A^{<\mathbb{N}}$ se puede definir con una fórmula infinita Σ_i^c (Π_i^c) con $i = 1, 2$.

Definición 5.3.5. Sean $\mathcal{A}$ una estructura, $\mathcal{M}$ una ω-presentación de $\mathcal{A}$ y $R \subseteq \mathbb{N} \times M^{<\mathbb{N}}$.

(i) Diremos que R es **Σ_i^c-definible con parámetros en $\mathcal{M}$** para $i = 1, 2$ si y sólo si existe una tupla $\overline{p} \in M^{<\mathbb{N}}$ y un conjunto computable enumerable de códigos de fórmulas $\varphi_i^{len(\overline{p})+j,\Sigma_i^c}(x_0, ..., x_{len(\overline{p})-1}, y_0, ..., y_{j-1})$ que son Σ_i^c tal que

$$R = \{(i, \overline{b}) \in \mathbb{N} \times M^{<\mathbb{N}} : \mathcal{M} \vDash \varphi_i^{len(\overline{p})+len(\overline{b}),\Sigma_i^c}[x_l \to p_l, y_j \to b_j : l < len(\overline{x}), j < len(\overline{y})]\}$$

(ii) Diremos que R es **Π_i^c-definible con parámetros en $\mathcal{M}$** para $i = 1, 2$ si y sólo si existe una tupla $\overline{p} \in M^{<\mathbb{N}}$ y un conjunto computable enumerable de códigos de fórmulas $\varphi_k^{len(\overline{p})+j,\Pi_i^c}(x_0, ..., x_{len(\overline{p})-1}, y_0, ..., y_{j-1})$ que son Π_i^c tal que

$$R = \{(i, \overline{b}) \in \mathbb{N} \times M^{<\mathbb{N}} : \mathcal{M} \vDash \varphi_i^{len(\overline{p})+len(\overline{b}),\Pi_i^c}[x_l \to p_l, y_j \to b_j : l < len(\overline{x}), j < len(\overline{y})]\}$$

La relación mencionada anteriormente es la que presentamos a continuación.

Proposición 5.3.3. *Sean $\mathcal{A}$ una estructura y $\mathcal{M}$ una ω-presentación de $\mathcal{A}$ y $R \subseteq \mathbb{N} \times M^{<\mathbb{N}}$ un conjunto. Si R es Σ_1^c-definible con parámetros en $\mathcal{M}$ entonces R es r.i.c.e. respecto a $\mathcal{M}$.*

Demostración. Supongamos que R es Σ_1^c-definible con parámetros en $\mathcal{M}$. Por consiguiente existen una tupla finita $\overline{p} \in M^{<\mathbb{N}}$ y un conjunto computable enumerable de códigos de fórmulas $\varphi_i^{len(\overline{p})+j,\Sigma_1^c}(x_0, ..., x_{len(\overline{p})-1}, y_0, ..., y_{j-1})$ tal que

$$R = \{(i, \overline{b}) \in \mathbb{N} \times M^{<\mathbb{N}} : \mathcal{M} \vDash \varphi_i^{len(\overline{p})+len(\overline{b}),\Sigma_i^c}[x_l \to p_l, y_j \to b_j : l < len(\overline{x}), j < len(\overline{y})]\}$$

Veamos que R es r.i.c.e. respecto a $\mathcal{M}$. Sea Φ_e la función inducida por el programa de la proposición 5,3,2. Sea $(\mathcal{B}, R)$ una ω-presentación de $(\mathcal{M}, R)$. Veamos que $R^{\mathcal{B}}$ es $D(\mathcal{B})$-computable enumerable. Para ello sea $n \in \omega$. A partir de n obtengamos de forma computable una tupla $(i, \overline{b}) \in \mathbb{N} \times B^{<\mathbb{N}}$. Buscamos de forma computable el código de la fórmula $\varphi_i^{len(\overline{p})+len(\overline{b}),\Sigma_1^c}$. Verificamos de forma computable si $\Phi_e^{D(\mathcal{B})}$ enumera en algún momento a $\boldsymbol{\psi}(\lceil \varphi_i^{len(\overline{b}),\Sigma_1^c} \rceil, \overline{b})$. En caso de que en algún momento obtengamos dicho elemento entonces es debe ocurrir que $\mathcal{M} \vDash \varphi_i^{len(\overline{p})+len(\overline{b}),\Sigma_1^c}[x_l \to p_l, y_j \to b_j : l < len(\overline{p}), j < len(\overline{b})]$. Finalmente devolvemos la tupla $(i, \overline{b})$ y el programa termina. En caso contrario el programa correrá indefinidamente buscando a $\boldsymbol{\psi}(\lceil \varphi_i^{len(\overline{b}),\Sigma_1^c} \rceil, \overline{b})$. Por lo tanto $R^{\mathcal{B}}$ es $D(\mathcal{B})$-computable enumerable. Finalmente esto nos permite concluir que R es r.i.c.e. respecto a $\mathcal{M}$.

□

La parte recíproca de la proposición anterior también es válida. La enunciamos a continuación.

Proposición 5.3.4. *Sean $\mathcal{A}$ una estructura y $\mathcal{M}$ una ω-presentación de $\mathcal{A}$ y $R \subseteq \mathbb{N} \times M^{<\mathbb{N}}$ un conjunto. Si R es r.i.c.e. respecto a $\mathcal{M}$ entonces $\mathcal{M}$ entonces R es Σ_1^c-definible con parámetros en $\mathcal{M}$.*

Demostración. Supongamos que R es r.i.c.e. respecto a $\mathcal{M}$. La idea es construir una ω-presentación $\mathcal{B}$ de $\mathcal{M}$ a partir del pull-back de una función sobreyectiva sobre la estructura $\mathcal{M}$. Dicha función se construirá recursivamente. En cada paso de su construcción trataremos de forzar que la ω-presentación de la función resultante no sea computable enumerable en $D(\mathcal{B})$. Como R es r.i.c.e. este intento fallará en un paso, lo cual nos llevará a dar una definición Σ_1^c con parámetros de R.

Definiremos una sucesión de funciones parciales $g_s : \operatorname{dom} g_s \subseteq \omega \to \omega$ tal que $g_s \subseteq g_{s+1}$. Sea $g_0 = \emptyset$ la función vacía. Supongamos que hemos definido g_s y definamos g_{s+1}.

Si $s + 1 = 2e + 1$ para algún $e \in \omega$ distinguimos de dos casos. Definimos $g_{s+1}(\operatorname{máx} \operatorname{dom} g_s + 1) = \operatorname{mín}\ \omega \setminus \operatorname{im} g_s$. Estos pasos nos asegurarán que la unión de las funciones g_s es sobreyectiva.

Ahora bien si $s + 1 = 2e$ para algún $e \in \omega$ entonces trataremos de hacer que $W_e^{D(\mathcal{B})}$ sea distinto a $R^{\mathcal{B}}$. Para ello buscamos una extensión de la tupla $\overline{g}_s = (g_s(0), ..., g_s(\operatorname{máx} \operatorname{dom} g_s))$, digamos $\overline{q} = (q_0, ..., q_n)$ con $n \in \omega$, en el conjunto

$$Q_e = \{\overline{q} \in M^{<\mathbb{N}} : \exists l, i, j_1, ..., j_l < len(\overline{q})((i, j_1, ..., j_l) \in W_e^{D_{\mathcal{A}}(\overline{q})} \ \&\ (i, q_{j_1}, ..., q_{j_l}) \notin R)\}$$

Si existe dicha tupla entonces definimos a g_{s+1} tal que su tupla asociada sea la tupla $\overline{q}$. En caso contrario $g_{s+1} = g_s$.

Definimos $g = \bigcup_{s \in \omega} g_s$. Notemos que g es una función sobreyectiva con dominio ω y codominio ω.

Notemos que en la etapa de construcción $s = 2e + 1$, si encontramos a la tupla $\overline{q}$ entonces como $\overline{q} \subseteq \overline{g}$ donde $\overline{g}$ es la sucesión asociada a g tenemos que $D_{\mathcal{M}}(\overline{q}) \subseteq D(\mathcal{B})$, lo cual implica que $(i, j_1, ..., j_l) \in W_e^{D_{\mathcal{M}}(\overline{q})} \subseteq W_e^{D(\mathcal{B})}$ y $(i, g(j_1), ..., g(j_l)) \notin R$. Por lo tanto $W_e^{D(\mathcal{B})}$ es distinto de $R^{\mathcal{B}}$. Como $R^{\mathcal{B}}$ debe ser $D(\mathcal{B})$-computable enumerable entonces se sigue que no siempre podemos encontrar una extensión $\overline{q}$ en Q_e para todo $e \in \omega$.

A partir de esta restricción, construiremos una definición Σ_1^c para R. Supongamos que $R^{\mathcal{B}} = W_e^{D(\mathcal{B})}$ para algún $e \in \omega$. Entonces para $s = 2e + 1$ tenemos que $\overline{g}_s$ no una extensión en Q_e. Veamos que R es Σ_1^c-definible con parámetros $\overline{g}_s$. Definamos al conjunto:

$$S = \{(i, q_{j_1}, ..., q_{j_l}) \in \mathbb{N} \times M^{<\mathbb{N}} : \begin{array}{l} \overline{q} \supseteq \overline{g}_s \in M^{<\mathbb{N}} \ \&\ l, i, j_1, ..., j_l < len(\overline{q}) \ \& \\ (i, j_1, ..., j_l) \in W_e^{D_{\mathcal{A}}(\overline{q})} \end{array} \}$$

Veamos que $R = S$. Sea $(i, \overline{a}) \in R$. Sean $j_1, ..., j_{len(\overline{a})} \in \omega$ tal que $\overline{a} = (g(j_1), ..., g(j_{len(\overline{a})}))$. Como $W_e^{D(\mathcal{B})} = R^{\mathcal{B}}$ y $(i, j_1, ..., j_{len(\overline{a})}) \in R^{\mathcal{B}}$ entonces $(i, j_1, ..., j_{len(\overline{a})}) \in W_e^{D(\mathcal{B})}$. Como $W_e^{D(\mathcal{B})} = \bigcup_{k \in \omega} W_e^{D_{\mathcal{M}}(g \restriction k)}$ tenemos que $(i, j_1, ..., j_{len(\overline{a})}) \in W_e^{D_{\mathcal{M}}(g \restriction k)}$ para algún $k \in \omega$. Como podemos tomar a k suficientemente grande podemos suponer sin pérdida de generalidad que $g \restriction k$ contiene a $\overline{g}_s$ y que $len(\overline{a}), i, j_1, ..., j_{len(\overline{a})} < len(g \restriction k)$. Por lo tanto concluimos que $(i, \overline{a}) \in S$.

Por otro lado sea $(i, \overline{a}) \in S$. Supongamos que $\overline{a}$ es de la forma $(q_{j_1}, ..., q_{j_l})$ para alguna $\overline{q} \supseteq \overline{g}_s$. Si $(i, \overline{a}) \notin R$ entonces $(i, q_{j_1}, ..., q_{j_l}) \notin R$, lo cual implica que $\overline{q} \in Q_e$, contradiciendo que no hay extensiones de $\overline{g}_s$ en Q_e. Por lo tanto $(i, \overline{a}) \in R$. Concluimos que $R = S$.

Para concluir la demostración basta con dar una definición Σ_1^c con parámetros $\overline{g}_s$ del conjunto S.

Definimos la fórmula que denotamos por $\varphi_i(\overline{g}_s, \overline{x})$ como la fórmula

$$\exists \overline{q} \supseteq \overline{g}_s \bigvee_{j_1, ..., j_{len(\overline{x})} < len(\overline{q})} ((q_{j_1}, ..., q_{j_{len(\overline{x})}}) = \overline{x} \ \wedge \ (i, j_1, ..., j_{len(\overline{x})}) \in W_e^{D_{\mathcal{M}}(\overline{q})})$$

Observemos que la fórmula anterior es tal que $(i, \overline{a}) \in S$ si y sólo si en $\mathcal{M}$ es cierta la fórmula $\varphi_i(\overline{g}_s, \overline{x})$ con $\overline{x}$ sustituido por $\overline{a}$.

La fórmula anterior la podemos ver como la fórmula siguiente:

$$\bigvee_{\substack{\sigma \in 2^{<\mathbb{N}}, \\ (i, j_1, ..., j_{len(\overline{x})}) \in W_e^{\sigma}, \\ \overline{q} \in \omega^{<\mathbb{N}}}} (\overline{q} \supseteq \overline{g}_s \ \wedge \ (q_{j_1}, ..., q_{j_{len(\overline{x})}}) = \overline{x} \ \wedge \ \sigma \subseteq D_{\mathcal{M}}(\overline{q}))$$

A pesar de ser una disyunción infinita, no es claro que la fórmula anterior sea una fórmula Σ_1^c. Para ello notemos lo siguiente.

Recordemos que las funciones $\boldsymbol{\varphi_3}$ y $\boldsymbol{\psi}$ son biyecciones computables de ω^3 a ω y $\omega^{<\mathbb{N}}$ a ω respectivamente. Definamos el conjunto

$$I_{i,\overline{x}} := \{n \in \omega : \boldsymbol{\varphi_3}(m_1, m_2, m_3) = n \ \& \ \boldsymbol{\psi}(m_1) \in 2^{<\mathbb{N}} \ \& \ len(\boldsymbol{\psi}(m_2)) = len(\overline{x}) \ \& \ (i, \boldsymbol{\psi}(m_2)) \in W_e^{\boldsymbol{\psi}(m_1)}\}$$

Si $\boldsymbol{\varphi_3}(m_1, m_2, m_3) = n$ entonces denotamos por $\overline{n}^1$ a $\boldsymbol{\psi}(m_1)$, $\overline{n}^2$ a $\boldsymbol{\psi}(m_2)$ y $\overline{n}^3$ a $\boldsymbol{\psi}(m_3)$. La fórmula $\varphi_i(\overline{g}_s, \overline{x})$ la podemos reescribir como

$$\bigvee_{n \in I_{i,\overline{x}}} (\overline{n}^3 \supseteq \overline{g}_s \ \wedge \ (\overline{n}^3_{\overline{n}^2_1}, ..., \overline{n}^3_{\overline{n}^2_{len(\overline{x})}}) = \overline{x} \ \wedge \ \overline{n}^1 \subseteq D_{\mathcal{M}}(\overline{n}^3))$$

donde $\overline{n}^1 \subseteq D_{\mathcal{M}}(\overline{n}^3)$ lo sustituimos por la fórmula $\varphi_{\overline{n}^1}(\overline{n}^3)$ y $\overline{n}^3 \supseteq \overline{g}_s$ por una sucesión finita de igualdades.

Observemos que $\varphi_i(\overline{g}_s, \overline{x})$ es una fórmula Σ_1^c y es tal que $(i, \overline{a}) \in S$ si y sólo si $\mathcal{M} \vDash \varphi_i^{\Sigma_1^c}[x_l \to a_l : l < len(\overline{a})]$. Como $R = S$, concluimos que R es Σ_1^c-definible con parámetros, lo cual concluye con la demostración.

□

Ahora bien, de entre los posibles conjuntos r.i.c.e. de una estructura tenemos a un conjunto r.i.c.e. importante. Este conjunto esencialmente agrega a una estructura las relaciones para saber que elementos satisfacen una determinada fórmula infinita Σ_1^c, lo cual no es posible determinarlo únicamente con el diagrama atómico de la estructura.

Definición 5.3.6. Sea $\mathcal{A}$ una estructura y $\mathcal{M}$ una ω-presentación de $\mathcal{A}$. La **relación de Kleene relativa a $\mathcal{M}$** se define como:

$$K^{\mathcal{M}} := \{(i, \overline{b}) \in \mathbb{N} \times M^{<\mathbb{N}} : \mathcal{M} \vDash \varphi_i^{len(\overline{b}), \Sigma_1^c}[x_l \to b_l : l < len(\overline{b})]\}$$

Del resultado anterior tenemos que $K^{\mathcal{M}}$ es r.i.c.e. respecto a $\mathcal{M}$.

Ahora bien, similar a la proposición anterior existe una relación entre los conjuntos Σ_2^c-definibles con parámetros con los conjuntos r.i.c.e. en una estructura $(\mathcal{M}, K^{\mathcal{M}})$.

Proposición 5.3.5. *Sean $\mathcal{A}$ una estructura y $\mathcal{M}$ una ω-presentación de $\mathcal{A}$. Si $R \subseteq \mathbb{N} \times M^{<\mathbb{N}}$ es un conjunto Σ_2^c-definible con parámetros en $\mathcal{M}$ entonces R es r.i.c.e. respecto a $(\mathcal{M}, K^{\mathcal{M}})$.*

Demostración. Supongamos que R es Σ_2^c-definible en $\mathcal{M}$ con parámetros. Entonces existen $\overline{p} \in M^{<\mathbb{N}}$ y un conjunto computable enumerable de fórmulas Σ_2^c, $\varphi_i^{len(\overline{p})+j, \Sigma_2^c}(x_0, ..., x_{len(\overline{p})-1}, y_0, ..., y_{j-1})$ tal que

$$R = \{(i, \overline{b}) \in \mathbb{N} \times M^{<\mathbb{N}} : \mathcal{M} \vDash \varphi_i^{len(\overline{p})+len(\overline{b}), \Sigma_2^c} \begin{array}{l}[x_l \to p_l, y_j \to b_j : \\ l < len(\overline{x}), j < len(\overline{y})]\end{array} \}$$

Ahora bien, por la proposición anterior basta demostrar que R es Σ_1^c-definible con parámetros en $(\mathcal{M}, K^M)$. Supongamos que una fórmula Σ_2^c, $\varphi_k^{len(\overline{p})+j, \Sigma_2^c}$ es de la forma:

$$\bigvee_{j \in J} \exists \overline{z}_j \bigwedge_{i \in I} \varphi_i^{\forall}(\overline{x}, \overline{y}, \overline{z}_j)$$

para algunos $\emptyset \neq I, J \subseteq \omega$.

Recordemos que si tenemos una fórmula libre de cuantificadores, el decir que no existen tuplas que satisfagan la negación de dicha fórmula en una estructura es

equivalente a decir que toda tupla satisface esa fórmula en dicha estructura. Por lo tanto la fórmula anterior satisface las mismas tuplas en $\mathcal{M}$ que la siguiente fórmula:

$$\bigvee_{j\in J} \exists \overline{y}_j \neg \bigvee_{i\in I} \varphi^{\exists}_{t(i)}(\overline{x}, \overline{y}, \overline{z}_j)$$

donde $t(i)$ se puede obtener de forma computable a partir de i.

Como $\bigvee\limits_{i\in I} \varphi^{\exists}_{t(i)}(\overline{x}, \overline{y}, \overline{z}_j)$ es una fórmula Σ^c_1 entonces por la definición de $K^{\mathcal{M}}$ debe ocurrir que esta es una fórmula libre de cuantificadores con el lenguaje de la estructura $(\mathcal{M}, K^{\mathcal{M}})$. Denotemos por $\varphi^{lc}_{s(j)}(\overline{x}, \overline{y}, \overline{z}_j)$ a dicha fórmula donde $s(j)$ es el número asociado, bajo una biyección computable de $\omega^{len(\overline{x})}$ a ω, de una tupla de la forma $(0, 1, \langle \overline{x} \rangle, e)$ donde e es tal que W_e es el conjunto de índices de las fórmulas existenciales con índices $t(i)$. Entonces tenemos que $\varphi_k^{\Sigma^c_2}$ satisface las mismas tuplas en $\mathcal{M}$ que la siguiente fórmula en $(\mathcal{M}, K^{\mathcal{M}})$:

$$\bigvee_{j\in J} \exists \overline{z}_j (\neg \varphi^{lc}_{s(j)}(\overline{x}, \overline{y}, \overline{z}_j))$$

Por observaciones anteriores la fórmula anterior es Σ^c_1 en el lenguaje de la estructura $(\mathcal{M}, K^{\mathcal{M}})$, denotemos por $\varphi^{len(\overline{p})+j, \Sigma^c_1}_{r(k)}$ a dicha fórmula. Ahora bien, dado que el conjunto de fórmulas $\varphi_k^{len(\overline{p})+j, \Sigma^c_2}$ es computable enumerable y podemos obtener de forma computable $r(k)$ a partir de k entonces el conjunto de códigos de fórmulas $\varphi^{len(\overline{p})+j, \Sigma^c_1}_{r(k)}$ también es computable enumerable.

Por último notemos que

$$R = \{(i, \overline{b}) \in \mathbb{N} \times M^{<\mathbb{N}} : (\mathcal{M}, K^{\mathcal{M}}) \vDash \varphi^{len(\overline{p})+len(\overline{b}), \Sigma^c_1}_{r(k)} \begin{array}{l}[x_l \to p_l, y_j \to b_j : \\ l < len(\overline{x}), j < len(\overline{y})]\end{array} \}$$

Por lo tanto R es Σ^c_1-definible en $(\mathcal{M}, K^{\mathcal{M}})$ con parámetros. Lo cual implica que R es r.i.c.e. en $(\mathcal{M}, K^{\mathcal{M}})$. □

Nos interesa tratar con conjuntos de tuplas cuyos elementos sean distintos entre sí digamos R tal que toda tupla que contenga una tupla de R deba pertenecer a R.

Definición 5.3.7. Sean $\mathcal{A}$ una estructura y A^* el conjunto de tuplas finitas de A sin repeticiones entre sus elementos.

(i) Diremos que $R \subseteq A^*$ es un **conjunto cerrado superiormente** si y sólo si para todo $\gamma \in R$, si $\sigma \supseteq \gamma$ entonces $\sigma \in R$.

(ii) Dado $R \subseteq A^*$, definimos la **cerradura superior** de R como el conjunto $\{\sigma \in A^* : \exists\, \gamma \in R\ (\gamma \subseteq \sigma)\}$

De las definiciones anteriores es sencillo notar que la cerradura superior de todo conjunto $R \subseteq A^*$ es un conjunto cerrado superiormente.

Recordemos que en la demostración del teorema 5,1,2 construimos una ω-presentación de una estructura a partir de una enumeración de esta. En ocasiones podemos hacer que tales enumeraciones satisfagan una condición específica a fin de conseguir que un pullback de la ω-presentación original satisfaga cierta condición en particular.

A continuación introduciremos enumeraciones que tienen segmentos iniciales que nos ayudan a determinar si toda extensión de dichos segmentos pertenece a un conjunto r.i.c.e. o no. De lo anterior se desprenden las siguientes definiciones.

Definición 5.3.8. Sean $\mathcal{A}$ una estructura y A^* el conjunto de tuplas finitas de A sin repeticiones entre sus elementos.

(i) Sea $R \subseteq A^*$ un conjunto cerrado superiormente. Decimos que una tupla $\gamma \in A^*$ **decide** R si y sólo si $\gamma \in R$ ó para todo $\sigma \in A^*$ si $\sigma \supseteq \gamma$ entonces $\sigma \notin R$.

(Ii) Sea $R \subseteq A^*$ un conjunto. Decimos que una tupla $\gamma \in A^*$ **decide** R si y sólo si γ decide la cerradura superior de R.

El tipo de enumeraciones que nos interesan son las que se muestran a continuación.

Definición 5.3.9. Sean $\mathcal{A}$ una estructura y A^* el conjunto de tuplas finitas de A sin repeticiones entre sus elementos.

(i) Una función $g : \omega \to A$ inyectiva es una **enumeración 1-genérica de $\mathcal{A}$** si y sólo si para todo conjunto r.i.c.e. R subconjunto de $\mathbb{N} \times A^*$ existe un segmento inicial de g que decide R.

(ii) Diremos que una función $g : \omega \to A$ inyectiva es **una enumeración 2-genérica de $\mathcal{A}$** si y sólo es una enumeración 1-genérica de $(\mathcal{A}, K^{\mathcal{A}})$.

Antes de continuar, mostraremos que la definición anterior no es vacía. Para ello mostraremos que toda ω-presentación de una estructura tiene una enumeración 1-genérica. Primero mostraremos que los conjuntos r.i.c.e. respecto a una estructura forman un conjunto computable enumerable.

Lema 5.3.1. *Sean $\mathcal{A}$ una estructura y $\mathcal{M}$ una ω-presentación de $\mathcal{A}$. Entonces el conjunto $\mathcal{R}$ definido como*

$$\{((e, \overline{a}), (i, \overline{b})) \in (\omega \times M^{<\mathbb{N}}) \times (\omega \times M^{<\mathbb{N}}) : \Phi_e(i)\downarrow \ \wedge \ (i, \overline{a}\overline{b}) \in K^{\mathcal{M}}\}$$

es $D(\mathcal{M})$-computable enumerable.

Demostración. Sea $e_0 \in \omega$ como en la proposición 5,3,2. Consideremos el siguiente programa: a partir de $n \in \omega$, obtengamos de forma computable una tupla de la forma (i, j). A partir de i obtengamos de forma computable una tupla de la forma $(e, \overline{a})$ y a partir de j una tupla de la forma $(i, \overline{b})$. Al final obtendremos una tupla $((e, \overline{a}), (i, \overline{b}))$ en $(\omega \times M^{<\mathbb{N}}) \times (\omega \times M^{<\mathbb{N}})$. Verificamos si $\Phi_e(i) \downarrow$. Observemos que en este paso la verificación puede llevarnos a nunca tener una conclusión de si $\Phi_e(i) \downarrow$ o no, lo cual produce que el programa que estamos definiendo nunca pueda detenerse y así nunca devolver un valor. En caso de que $\Phi_e(i) \downarrow$, entonces proseguimos con el siguiente paso.

Ahora verificamos si $\mathcal{M} \vDash \varphi_i^{len(\overline{a})+len(\overline{b})}[x_l \to a_l, y_j \to b_j : l < len(\overline{x}), j < len(\overline{y})]$ verificando si $\boldsymbol{\psi}(i, \overline{a}\overline{b}) \in W_{e_0}^{D(\mathcal{M})}$. Es decir vemos si $\Phi_{e_0}(\boldsymbol{\psi}(i, \overline{a}\overline{b})) \downarrow$ con la ejecución directa de dicho programa. Nuevamente como en el paso anterior no sabemos a ciencia cierta si Φ_{e_0} se detendrá en dicha entrada, lo cual hara que el programa que estamos definiendo no se detenga y así nunca devuelva un valor. En caso de que se detenga sabremos que $\boldsymbol{\psi}(i, \overline{a}\overline{b}) \in W_{e_0}^{D(\mathcal{M})}$ y por lo tanto que $\mathcal{M} \vDash \varphi_i^{len(\overline{a})+len(\overline{b})}[x_l \to a_l, y_j \to b_j : l < len(\overline{x}), j < len(\overline{y})]$. Si llegamos a este paso únicamente bastará con devolver la tupla $((e, \overline{a}), (i, \overline{b}))$, la cual por todo lo verificado anteriormente tenemos que está en $\mathcal{R}$. El programa anterior define una función parcial $D(\mathcal{M})$-parcialmente computable cuya imagen es $\mathcal{R}$.

Por lo tanto $\mathcal{R}$ es $D(\mathcal{M})$-computable enumerable. □

Como para todo conjunto r.i.c.e. R respecto a una ω-presentación $\mathcal{M}$ existe una tupla $\overline{p}$ y un conjunto c.e. de fórmulas φ_i^{j,Σ_1^c}, digamos W_e, tal que

$$R = \{(i, \overline{b}) \in \mathbb{N} \times M^{<\mathbb{N}} : \mathcal{M} \vDash \varphi_i^{len(\overline{p})+len(\overline{b}),\Sigma_1^c}\ \begin{matrix}[x_l \to p_l, y_j \to b_j \\ l < len(\overline{x}), j < len(\overline{y})]\end{matrix}\ \}$$

tenemos que podemos obtener los elementos de R a partir de los elementos de $\mathcal{R}$ buscando tuplas de la forma $((e, \overline{p}), (i, \overline{b}))$ en $\mathcal{R}$ de forma $D(\mathcal{M})$-computable como en el lema anterior. Como en el lema anterior $(e, \overline{p})$ son definidos a partir de un número natural i y dicho número no cambia por todos los elementos de R entonces diremos que R es el i-ésimo conjunto r.i.c.e. respecto a $\mathcal{M}$ y lo denotaremos también por R_i. También como $(i, \overline{b})$ es definido por un número natural j entonces diremos únicamente que $\overline{b}$ es el j-ésimo elemento de R, o bien el j-ésimo elemento de R_i.

Por todo lo mencionado, denotamos por $\{R_n : n \in \omega\}$ a una enumeración $D(\mathcal{M})$-computable de los conjuntos r.i.c.e. respecto a $\mathcal{M}$ en el entendido de que podemos obtener, si existe, de forma $D(\mathcal{M})$-computable el j-ésimo elemento de R_i para cualesquiera $i, j \in \omega$.

Una observación similar a la anterior es considerar una modificación del conjunto

$\mathcal{R}$ como el siguiente conjunto

$$\mathcal{R}^* := \{((e,\overline{a}),(i,\overline{b})) \in (\omega \times M^{<\mathbb{N}}) \times (\omega \times M^{<\mathbb{N}}) : \begin{array}{l} \forall i,j < len(\overline{b})(\ b_i \neq b_j) \\ \wedge\ \Phi_e(i)\downarrow\ \ \wedge\ \ (i,\overline{a}\overline{b}) \in K^{\mathcal{M}} \end{array}\ \}$$

Dicho conjunto también es $D(\mathcal{M})$-computable. Esto lo podemos ver si añadimos un paso más a la construcción del programa del lema anterior en el cual verificamos que la tupla obtenida $(i,\overline{b})$ tiene todos sus elementos distintos. En caso de que no se cumpla este requisito entonces el programa corre indefinidamente. En caso de que sí el programa continúa tal como se describe en la demostración.

Por lo observado anteriormente, tenemos una enumeración de los conjuntos r.i.c.e. cuyos elementos son todos distintos y de sus elementos. Denotamos por $\{R_n : n \in \omega\}$ a una enumeración $D(\mathcal{M})$-computable de los conjuntos r.i.c.e. cuyos elementos son todos distintos respecto a $\mathcal{M}$ en el entendido de que podemos obtener, si existe, de forma $D(\mathcal{M})$-computable el j-ésimo elemento de R_i para cualesquiera $i,j \in \omega$.

Proposición 5.3.6. *Sean $\mathcal{A}$ una estructura y $\mathcal{M}$ una ω-presentación de $\mathcal{A}$. Entonces existe una enumeración 1-genérica sobreyectiva de $\mathcal{M}$.*

Demostración. Construiremos una función sobreyectiva $g : \omega \to M$ que satisfaga las condiciones para ser 1-genérica. Para ello construiremos una sucesión de funciones parciales que satisfagan ciertos requerimientos y tomaremos la unión de todas ellas para obtener la función g que buscamos. Sea $\{R_n : n \in \omega\}$ una enumeración de los conjuntos r.i.c.e., cuyos elementos son todos distintos, respecto de $\mathcal{M}$. La idea es hacer que la construcción de la función g en el paso s decida al conjunto r.i.c.e. R_s.

Para ello definimos $g_0 : \text{dom } g_0 \subseteq \omega \to \omega$ tal que dom g_0 es un segmento inicial de ω y $(g_0(0),...,g_0(\text{máx dom } g_0)) = \overline{q}$ donde $\overline{q}$ es el primer elemento enumerado en las tuplas finitas en M^* de R_0 en caso de que exista. En caso contrario definimos $g_0 = \emptyset$. Supongamos que hemos definido g_s tal que dom g_s es un segmento inicial de ω. Definamos g_{s+1} de manera similar a como definimos g_0. Si existe $\overline{q} \supseteq \overline{g}_s$ tal que $\overline{q}$ es el primer elemento enumerado en las tuplas finitas de $M*$ de R_{s+1} entonces sea $g_{s+1} : \text{dom } g_{s+1} : \omega \to \omega$ donde $g_{s+1} = \overline{q}$. En caso contrario definimos $g_{s+1} = g_s$.

Por definición tenemos que para cada $s \in \omega$, $g_s \subseteq g_{s+1}$. Por lo tanto $g := \bigcup_{s\in\omega} g_s$ es una función. Notemos que g es una función inyectiva ya que estamos tomando elementos de conjuntos r.i.c.e. cuyos elementos son distintos entre sí.

Veamos que g es 1-genérica. Para ello sea R un conjunto r.i.c.e. respecto de $\mathcal{M}$ cuyos elementos de sus tuplas son distintos entre sí. Supongamos que $R = R_s$ para algún $s \in \omega$. Sea $k = \text{máx dom } g_s$. Veamos que $g \upharpoonright k+1 = (g(0),...,g(k))$ decide la cerradura superior de R. Para ello basta notar que si $g \upharpoonright k+1$ no está en la

cerradura superior de R entonces si $\sigma \supseteq g \upharpoonright k+1$ tenemos que σ no está en la cerradura superior de R.

En efecto, supongamos lo contrario, es decir que $\sigma \supseteq g \upharpoonright k+1$ y que σ está en la cerradura superior de R. Entonces podemos concluir que $R_s \neq \emptyset$. Por lo tanto en el paso s debe ser enumerada una tupla γ en R_s. DIcha tupla no puede estar contenida en $g \upharpoonright k+1$ ya que $g \upharpoonright k+1$ no está en la cerradura superior de R. Por lo tanto dicha tupla extiende a $g \upharpoonright k+1 = g_s$. Por la construcción de las funciones g_s tenemos que $g \upharpoonright k+1 = g_s = \gamma$. Por lo tanto $g \upharpoonright k+1 \in R$, lo cual implica que $g \upharpoonright k+1$ está en la cerradura superior de R, lo cual es una contradicción al supuesto de que $g \upharpoonright k+1$ no estaba en dicho conjunto. Por lo tanto σ no está en la cerradura superior de R. De esta forma concluimos que $g \upharpoonright k+1$ decide la cerradura superior de R. Por lo tanto podemos concluir que g es 1-genérica.

Veamos que g es sobreyectiva. Sea $m \in M$. Definimos $D_m := \{(1, \overline{p}) \in \mathbb{N} \times M^* : \exists i < len(\overline{p})(p_i = m)\} \subseteq M^*$ donde M^* es el conjunto de tuplas cuyos elementos son distintos entre sí. Afirmamos que D_m es r.i.c.e. respecto a $\mathcal{M}$. En efecto, notemos que si tomamos cualquier otra ω-presentación $\mathcal{B}$ de $\mathcal{M}$, generamos de forma computable una tupla finita con todos sus elementos disitintos entre sí y verificamos si alguno de dichos elementos es igual a m. En caso de que alguna de las verificaciones falle, el programa entra en un ciclo que no termina. En otro caso, enumeramos dicha tupla. Por lo tanto $(D_m)^{\mathcal{B}}$ es $D(\mathcal{B})$-computable enumerable. Por lo tanto concluimos que D_m es r.i.c.e. respecto de $\mathcal{M}$.

Luego, $D_m = R_s$ para algún $s \in \omega$. Como g es 1-genérica entonces existe un segmento inicial de g, digamos $g \upharpoonright k$, que decide a D_m. Como podemos extender a $g \upharpoonright k$ a una tupla que tenga en una coordenada a m entonces es falso que toda extensión de $g \upharpoonright k$ no esté en la cerradura superior de D_m. Por lo tanto $g \upharpoonright k+1$ está en la cerradura superior de D_m. Lo cual implica que la tupla $g \upharpoonright k+1$ ya enumeró al elemento m. Como $g \upharpoonright k+1 \subseteq g$ entonces tenemos que existe $l \in \omega$ tal que $g(l) = m$. Por lo tanto g es sobreyectiva.

□

Retomemos ahora la cuestión que nos trajo hasta aquí; la versión no uniforme de un teorema que caracterice la existencia de isomorfismos computables entre las presentaciones de una estructura. A continuación damos una par de definiciones para esclarecer la noción que estamos comentando.

Definición 5.3.10. Una estructura computable $\mathcal{A}$ es **computablemente categórica** si y sólo si existe un isomorfismo computable entre cualesquiera dos estructuras computables de $\mathcal{A}$.

Y la versión relativa a cualquier oráculo $X \subseteq \omega$ es la siguiente.

Definición 5.3.11. *(i)* Dados $X \subseteq \omega$ y $\mathcal{A}$ una estructura X-computable, decimos que $\mathcal{A}$ es **X-computablemente categórica** si y sólo si existe un isomorfismo X-computable entre cualesquiera dos estructuras X-computables isomorfas de $\mathcal{A}$.

(ii) Si $\mathcal{A}$ es computable entonces diremos que es **relativa y computablemente categórica** si y sólo si $\mathcal{A}$ es X-computablemente categórica para cada $X \subseteq \omega$.

Finalmente presentamos la caracterización de la versión no uniforme del teorema 5,2,1.

Teorema 5.3.1. *Sea $\mathcal{A}$ una estructura computable. Las siguientes son equivalentes:*

(i) *$\mathcal{A}$ es relativa y computablemente categórica*

(ii) *Existe $\overline{a} \in A^{<\mathbb{N}}$ tal que $(\mathcal{A}, \overline{a})$ es uniforme y computablemente categórica.*

(iii) *Existen $\overline{a} \in A^{<\mathbb{N}}$ y una familia de Scott computable enumerable para $(\mathcal{A}, \overline{a})$ que consiste de fórmulas existenciales.*

Demostración. Por el teorema 5,2,1 sabemos que *(ii)* es equivalente a *(iii)*.

Veamos que el inciso *(ii)* implica el inciso *(i)*. Sea $X \subseteq \omega$ y veamos que $\mathcal{A}$ es X-computablemente categórica. Para ello sean $\mathcal{B}$ y $\mathcal{C}$ dos estructuras X-computables isomorfas a $\mathcal{A}$. Ahora bien por el teorema 5,2,1 tenemos que $(\mathcal{A}, \overline{a})$ es uniforme, relativa y computablemente categórica. Sea $e \in \omega$ tal que Φ_e hace a $\mathcal{A}$ uniforme, relativa y computablemente categórica. Sea e_0 el número del programa cuya función inducida es $\Phi_{e_0}^{D(\mathcal{B})\oplus D(\mathcal{C})}$ y que es igual a la función $(\Phi_e^{D(\mathcal{C},\overline{a}^{\mathcal{C}})})^{-1} \circ \Phi_e^{D(\mathcal{B},\overline{a}^{\mathcal{B}})}$ con la modificación de que si en algún momento al usar $D(\mathcal{B}, \overline{a}^{\mathcal{B}})$ preguntamos por el índice de una fórmula que tenga a alguno de los elementos constantes de $\overline{a}^{\mathcal{B}}$ entonces preguntamos por el índice de la fórmula que tiene la misma forma que la anterior con la excepción de que cada ocurrencia de cada constante la cambiamos por una variable cuyo índice es el valor de dicha constante. Si dicha variable ya está siendo utilizada por un cuantificador, cambiamos la variable cuantificada por una que no haya sido utilizada. Hacemos lo mismo si utilizamos $(D(\mathcal{C}), \overline{a}^{\mathcal{C}})$. Dentro de esta sustitución, los valores que obtengamos de consultar los índices de fórmulas que tengan constantes deben ser los mismos que obtengamos tras consultar los índices de las fórmulas obtenidas tras la sustitución.

Por consiguiente los resultados obtenidos por este programa con la modificación debe ser igual que $(\Phi_e^{D(\mathcal{C},\overline{a}^{\mathcal{C}})})^{-1} \circ \Phi_e^{D(\mathcal{B},\overline{a}^{\mathcal{B}})}$. Como este último es un isomorfismo de $(\mathcal{B}, \overline{a}^{\mathcal{B}})$ a $(\mathcal{C}, \overline{a}^{\mathcal{C}})$ entonces $\Phi_{e_0}^{D(\mathcal{B})\oplus D(\mathcal{C})}$ debe ser un isomorfismo de $\mathcal{B}$ a $\mathcal{C}$.

Por lo tanto, nuestro nuevo programa sólo utiliza como oráculos a $D(\mathcal{B})$ y $D(\mathcal{C})$, es decir al conjunto $D(\mathcal{B}) \oplus D(\mathcal{C})$. Como $\mathcal{B}$ y $\mathcal{C}$ son X-computables, tenemos que

el isomorfismo $\Phi_{e_0}^{D(\mathcal{B})\oplus D(\mathcal{C})}$ es X-computable. Esto nos permite concluir que $\mathcal{A}$ es X-computablemente categórica. Por lo tanto $\mathcal{A}$ es relativa y computablemente categórica.

Ahora demostremos que el inciso *(i)* implica el inciso *(iii)*. Para ello supongamos que $\mathcal{A}$ es relativa y computablemente categórica. Nuestro objetivo será el encontrar una tupla $\overline{a} \in A^{<\mathbb{N}}$ y construir una familia de Scott computable enumerable de fórmulas existenciales para $(\mathcal{A}, \overline{a})$.

Sea $g : \omega \to A$ una enumeración 2-genérica de $\mathcal{A}$, es decir una enumeración 1-genérica de $(\mathcal{A}, K^{\mathcal{A}})$, la cual existe por la proposición 5,3,6. Definimos $\mathcal{B} := g^{-1}(\mathcal{A})$. Tenemos que $\mathcal{B}$ es isomorfa a $\mathcal{A}$. Como $\mathcal{A}$ es relativa y computablemente categórica y $\mathcal{A}$ y $\mathcal{B}$ son $D(\mathcal{B})$-computables, entonces existe un $e \in \omega$ tal que $\Phi_e^{D(\mathcal{B})}$ es un isomorfismo $D(\mathcal{B})$-computable de $\mathcal{B}$ a $\mathcal{A}$.

Afirmación. Afirmamos que existe $\overline{p} \in A^{<\mathbb{N}}$ tal que $\overline{p} \subseteq \overline{g}$ y para cualquier $\overline{q} \in A^{<\mathbb{N}}$ tal que $\overline{p} \subseteq \overline{q}$, se tiene que $\overline{q}$ puede extenderse a una enumeración g' que induce una estructura $\mathcal{B}' = (g')^{-1}(\mathcal{A})$ tal que $\Phi_e^{D(\mathcal{B}')}$ es un isomorfismo de $\mathcal{B}'$ a $\mathcal{A}$.

Para demostrar esto, definimos

$$Q_1 := \{(i, \overline{q}) \in \mathbb{N} \times A^* : i = len(\overline{q}) \ \& \ \exists n < len(\overline{q})(\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright n \downarrow \ \& \\ \& \ D_{\mathcal{A}}(\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright n) \neq D_{\mathcal{A}}(\overline{q} \upharpoonright n))\}$$

Veamos que Q_1 es r.i.c.e. respecto de $\mathcal{A}$. Para ello sea $\mathcal{B}$ una ω-presentación de $\mathcal{A}$ y veamos que $Q_1^{\mathcal{B}}$ es $D(\mathcal{B})$-computable enumerable. Dado $n \in \omega$, obtenemos de forma computable una tupla $(i, \overline{q})$ y verificamos si $i = len(\overline{q})$ y si $\overline{q}$ tiene todos sus elementos distintos entre sí. En caso de que alguna de las verificaciones sea falsa, el programa entra en un ciclo infinito y no devuelve ninguna tupla. En caso contrario proseguimos con el siguiente paso. Ahora verificamos si $\Phi_e^{D_{\mathcal{B}}(\overline{q})} \downarrow n$ para algún $n < len(\overline{q})$. Para esto verificamos simultáneamente para cada $n < len(\overline{q})$ si $\Phi_e^{D_{\mathcal{B}}(\overline{q})}(n) \downarrow$, mediante la ejecución directa del programa de dicha función en la entrada n para cada $n < len(\overline{q})$. Notemos que en este paso podemos no detenernos ya que alguna de las ejecuciones anteriores puede no detenerse. En caso de que todas se detengan, proseguimos con el siguiente paso. En este paso ya tenemos la certeza de que $\Phi_e^{D_{\mathcal{B}}(\overline{q})} \downarrow n$ para algún $n < len(\overline{q})$. Entonces ahora, utilizando a $D(\mathcal{B})$ como oráculo, calculamos $D_{\mathcal{B}}(\Phi_e^{D_{\mathcal{B}}(\overline{q})} \upharpoonright n \downarrow)$ y $D_{\mathcal{B}}(\overline{q} \upharpoonright n)$ y los comparamos. Si son iguales entonces el programa entra en un ciclo infinito y no devuelve ninguna tupla. En caso contrario el programa devuelve la tupla $(i, \overline{q})$ y termina. La función inducida por este programa es $D(\mathcal{B})$-computable y su imagen es Q_1. Por lo tanto Q_1 es $D(\mathcal{B})$-computable enumerable. Por lo tanto concluimos que Q_1 es r.i.c.e. respecto de $\mathcal{A}$. Más aún podemos concluir que Q_1 es r.i.c.e. respecto de $(\mathcal{A}, K^{\mathcal{A}})$ ya que el conjunto $K^{\mathcal{A}}$ no fue necesario en el argumento anterior.

Como g es 2-genérica, entonces existe $k_1 \in \mathbb{N}$ tal que $g \upharpoonright k_1$ decide a Q_1. Afirmamos que $g \upharpoonright k_1$ no es una tupla $\overline{q}$ tal que $(i, \overline{q}) \in Q_1$. Por la construcción de g tenemos que $D(\mathcal{B})$ es la unión de los $D_{\mathcal{A}}(g \upharpoonright k)$ con $k \in \omega$. Aunque es posible que $\Phi_e^{D_{\mathcal{A}}(g\upharpoonright k_1)} \upharpoonright n \downarrow$ no puede ocurrir que $D_{\mathcal{A}}(\Phi_e^{D_{\mathcal{A}}(g\upharpoonright k_1)} \upharpoonright n) \neq D_{\mathcal{A}}((g \upharpoonright k_1) \upharpoonright n)$ ya que tanto $\Phi_e^{D(\mathcal{B})}$ como g son isomorfismos de $\mathcal{B}$ a $\mathcal{A}$, por lo que $\Phi_e^{D_{\mathcal{A}}(g\upharpoonright k_1)} \upharpoonright n$ y $(g \upharpoonright k_1) \upharpoonright n$ son automórficos. Por consiguiente $(i, g \upharpoonright k_1) \notin Q_1$ donde $i = len(g \upharpoonright k_1)$. Como $g \upharpoonright k_1$ decide a Q_1 entonces toda extensión γ de $g \upharpoonright k_1$ es tal que $(i, \gamma) \notin Q_1$ donde $i = len(\gamma)$.

Ahora consideremos al conjunto

$$Q_2 := \{(i, \overline{q}) \in \mathbb{N} \times A^* : i = len(\overline{q}) \ \wedge \ \exists n \in \mathbb{N}(\forall \overline{r} \in A^*(\overline{q} \subseteq \overline{r} \Rightarrow \Phi_e^{D_{\mathcal{A}}(\overline{r})}(n) \uparrow))\}$$

Veamos que Q_2 es un conjunto Σ_2^c-definible con parámetros. Para ello notemos que el complemento del conjunto

$$Q_2^* := \{(n, \overline{q}) \in \omega \times A^* : \forall \overline{r} \in A^*(\overline{q} \subseteq \overline{r} \ \wedge \ \Phi_e^{D_{\mathcal{A}}(\overline{r})}(n) \uparrow)\}$$

es r.i.c.e. respecto de $\mathcal{A}$, por lo tanto es Σ_1^c-definible con parámetros. Por lo tanto Q_2^* es Π_1^c-definible con parámetros. Sean $\overline{s} \in A^{\mathbb{N}}$ y $\varphi_i^{len(\overline{s})+j,\Pi_1^c}(y_0, ..., y_{\overline{s}-1}, x_0, ..., x_{j-1})$ las fórmulas que hacen a Q_2^* un conjunto Π_1^c-definible con parámetros.

Ahora observemos que $(i, \overline{q}) \in Q_2$ si y sólo si $\mathcal{A} \vDash \varphi_i^{len(\overline{s})+j,\Sigma_2^c}[s_j \to y_j, x_l \to q_l : j < len(\overline{y}), l < len(\overline{x})]$ donde $\varphi_i^{len(\overline{s})+j,\Sigma_2^c}$ es la i-ésima fórmula del conjunto de códigos de fórmulas Σ_2^c de la forma

$$i = len(\overline{x}) \ \wedge \ \bigvee_{n\in\mathbb{N}} \exists z \ (\varphi_n^{len(\overline{s})+j,\Pi_1^c}(\overline{y}, \overline{x}))$$

De lo mencionado anteriormente, tenemos que Q_2 es Σ_2^c-definible con parámetros. Por lo tanto Q_2 es r.i.c.e. respecto de $(\mathcal{A}, K^{\mathcal{A}})$. Por la definición de g tenemos que existe $k_2 \in \omega$ tal que $g \upharpoonright k_2$ decide a Q_2. Si ocurriera que $(i, g \upharpoonright k_2) \in Q_2$ entonces existe $n \in \mathbb{N}$ tal que toda extensión $\overline{r}$ de $g \upharpoonright k_2$ implica que $\Phi_e^{D_{\mathcal{A}}(\overline{r})}(n) \uparrow$. En particular las extensiones en g de $g \upharpoonright k_2$ satisfacen la propiedad anterior. Esto implica que $\Phi_e^{D(\mathcal{B})}(n) \uparrow$, lo cual contradice que $\Phi_e^{D(\mathcal{B})}$ es una función total. Por lo tanto $(i, g \upharpoonright k_2) \notin Q_2$ y toda extensión γ de $g \upharpoonright k_2$ es tal que $(i, \gamma) \notin Q_2$ con $i = len(g \upharpoonright k_2)$.

Definimos $\overline{p} = g \upharpoonright k$ donde k es el menor número natural tal que $g \upharpoonright k_1, g \upharpoonright k_2 \subseteq g \upharpoonright k$. Tenemos que $\overline{p} \subseteq \overline{g}$. Tenemos también que toda extensión $\overline{q}$ de $\overline{p}$ es tal que $(i, \overline{q}) \notin Q_1$ y $(i, \overline{q}) \notin Q_2$ con $i = len(\overline{q})$.

Ahora demostremos la afirmación que hemos mencionado más atrás. Para ello sea $\overline{q} \in A^{<\mathbb{N}}$ tal que $\overline{p} \subseteq \overline{q}$. Veamos que podemos extender a $\overline{q}$ a una enumeración $g' : \omega \to A$ tal que g' induce una estructura $\mathcal{B}' = (g')^{-1}(\mathcal{A})$ tal que $\Phi_e^{D(\mathcal{B}')}$ es un

isomorfismo de $\mathcal{B}'$ a $\mathcal{A}$.

Como $(i, \overline{q}) \notin Q_2$ con $i = len(\overline{q})$, para $n = 0$ existe $\overline{r}'_0 \in A^*$ tal que $\overline{q} \subseteq \overline{r}'_0$ y $\Phi_e^{D_{\mathcal{A}}(\overline{r}'_0)}(n) \downarrow$. Si 0 está en la tupla $\overline{r}'_0$ entonces definimos $\overline{r}_0 = \overline{r}'_0$. En otro caso definimos $\overline{r}_0 = \overline{r}'_0 0$. Como $\overline{r}_0$ extiende a $\overline{p}$ podemos aplicar el mismo proceso para obtener una tupla $\overline{r}_1 \in A^*$ tal que $\overline{q} \subseteq \overline{r}_0 \subseteq \overline{r}_1$, $\overline{r}_1$ tiene al número 1 y $\Phi_e^{D_{\mathcal{A}}(\overline{r}_1)}(n) \downarrow$ donde $n = 1$. Siguiendo este proceso construimos una sucesión de tuplas $\{\overline{r}_n : n \in \omega\}$ tal que para cada $n \in \omega$, n está en la tupla $\overline{r}_n$, $\overline{r}_n \subseteq \overline{r}_{n+1}$ y $\Phi_e^{D_{\mathcal{A}}(\overline{r}_n)}(n) \downarrow$. Definimos g' tal que su tupla asociada $\overline{g'}$ es tal que $\overline{g'} = \bigcup_{n \in \omega} \overline{r}_n$. Como para cada $n \in \omega$, $\overline{r}_n \subseteq \overline{r}_{n+1}$ entonces g' es una función. Como tenemos una sucesión infinita de $\overline{r}_n$ entonces dom $g' = \omega$. Dado que todos los elementos de $\overline{r}_n$ son distintos entre sí entonces g' es una función inyectiva. Y como $\overline{r}_n$ tiene a n entonces g' es una función sobreyectiva. Por lo tanto g' es una función biyectiva.

Definimos $\mathcal{B}' = (g')^{-1}(\mathcal{A})$. Ahora bien, para cada $n \in \omega$, como $\Phi_e^{D_{\mathcal{A}}(\overline{r}_n)}(n) \downarrow$ entonces $\Phi_e^{D(\mathcal{B}')}(n) \downarrow$. Por lo tanto $\Phi_e^{D(\mathcal{B}')}$ es una función total.

También como $(i, \overline{q}) \notin Q_1$ con $i = len(\overline{q})$, más aún para cada $n \in \omega$, $(i, \overline{r}_n) \notin Q_1$ con $i = len(\overline{r}_n)$ entonces para todo $m < len(\overline{r}_n)$, $\Phi_e^{D_{\mathcal{A}}(\overline{r}_n)} \upharpoonright m \downarrow$ implica que

$$D_{\mathcal{A}}(\Phi_e^{D_{\mathcal{A}}(\overline{r}_n)} \upharpoonright m) = D_{\mathcal{A}}(\overline{r}_n \upharpoonright m)$$

Sean R_i un símbolo relacional en el lenguaje de $\mathcal{A}$ (que también es el de $\mathcal{B}'$) y una tupla $(b_1, ..., b_{a_R(i)}) \in B'^{<\mathbb{N}}$. Sea $m \in \omega$ tal que $m > b_j$ para cada $j \leq a_R(i)$, $m > i$ y $m > a_R(i)$. Sea $k \in \omega$ tal que $i < m < len(\overline{r}_k)$ y $\Phi_e^{D_{\mathcal{A}}(\overline{r}_k)} \upharpoonright m \downarrow$, es decir $\Phi_e^{D_{\mathcal{A}}(\overline{r}_k)}(l) \downarrow$ para cada $l < m$, en particular para cada b_j con $j \leq a_R(i)$. Por lo tanto $D_{\mathcal{A}}(\Phi_e^{D_{\mathcal{A}}(\overline{r}_k)} \upharpoonright m) = D_{\mathcal{A}}(\overline{r}_k \upharpoonright m)$.

Por la definición de g como unión de los $\overline{r}_n$ y como $m > b_j$ para cada $j \leq a_R(i)$, tenemos que en $\overline{r}_k \upharpoonright m$ ya aparecen los valores de $g(b_j)$ para cada $j \leq a_R(i)$. Como $i < m$ la relación R_i está considerada en ambos diagramas $D_{\mathcal{A}}(\Phi_e^{D_{\mathcal{A}}(\overline{r}_k)} \upharpoonright m)$ y $D_{\mathcal{A}}(\overline{r}_k \upharpoonright m)$. Como $m > b_j$ para cada $j \leq a_R(i)$ entonces toda variable x_{b_j} se considera para ser parte de las fórmulas en los diagramas mencionados anteriormente. Y también dado que $m > a_R(i)$ entonces tenemos en cuenta combinaciones de variables de longitud $a_R(i)$ que componen fórmulas en los diagramas anteriores. Por lo tanto la fórmula, digamos de índice s, de la forma $R_i(x_{b_1}, ..., x_{b_{a_R(i)}})$ está considerada en ambos diagramas. Por la igualdad de dichos diagramas, tenemos que dichas fórmulas o bien se satisfacen en $\mathcal{A}$ simultáneamente tras sustituir sus variables por los elementos en las posiciones b_j de $\Phi_e^{D_{\mathcal{A}}(\overline{r}_k)} \upharpoonright m$ y $\overline{r}_k \upharpoonright m$ o bien esto no ocurre al mismo tiempo.

Por lo tanto, $(\Phi_e^{D_{\mathcal{A}}(\overline{r}_k)}(b_1), ..., \Phi_e^{D_{\mathcal{A}}(\overline{r}_k)}(b_{a_R(i)})) \in R_i^{\mathcal{A}}$ si y sólo si $(g'(b_1), ..., g'(b_{a_R(j)})) \in R_i^{\mathcal{A}}$. Por lo tanto $(\Phi_e^{D(\mathcal{B}')}(b_1), ..., \Phi_e^{D(\mathcal{B}')}(b_{a_R(i)})) \in R_i^{\mathcal{A}}$ si y sólo si $(g'(b_1), ..., g'(b_{a_R(j)})) \in R_i^{\mathcal{A}}$. Como g' es un isomorfismo de $\mathcal{B}'$ a $\mathcal{A}$ entonces $(g'(b_1), ..., g'(b_{a_R(j)})) \in R_i^{\mathcal{A}}$ si y sólo si $(b_1, .., b_{a_R(j)}) \in R_i^{\mathcal{B}'}$. Por lo tanto $(\Phi_e^{D(\mathcal{B}')}(b_1), ..., \Phi_e^{D(\mathcal{B}')}(b_{a_R(i)})) \in R_i^{\mathcal{A}}$ si y sólo si $(b_1, .., b_{a_R(j)}) \in R_i^{\mathcal{B}'}$. Concluimos que $\Phi_e^{D(\mathcal{B}')}$ es un isomorfismo de $\mathcal{B}'$ a $\mathcal{A}$. Esta último termina la demostración de la afirmación que hemos hecho en unos párrafos anteriores.

Una segunda afirmación que nos ayudará a construir la familia de Scott de fórmulas existenciales es la siguiente.

Afirmación. Para cualesquiera $\overline{n} = (n_1, ..., n_l), \overline{q}, \overline{r} \in A^{<\mathbb{N}}$ si $l < len(\overline{q}), len(\overline{r})$, para todo $n_i < len(\overline{q}), len(\overline{r})$, $\overline{q}, \overline{r} \supseteq \overline{p}$ y $\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright \overline{n} \downarrow$ entonces $D_{\mathcal{A}}(\overline{q}) = D_{\mathcal{A}}(\overline{r})$ implica que $(\mathcal{A}, \overline{q} \upharpoonright \overline{n}) \cong (\mathcal{A}, \overline{r} \upharpoonright \overline{n})$.

Sean $\overline{n}, \overline{q}, \overline{r} \in A^{<\mathbb{N}}$ como en la hipótesis de la afirmación. Supongamos que $D_{\mathcal{A}}(\overline{q}) = D_{\mathcal{A}}(\overline{r})$.

Entonces para $\overline{q}$, sea g' la extensión de $\overline{q}$ dada por la afirmación anterior y definamos $\mathcal{B}' = g^{-1}(\mathcal{A})$. Por lo tanto $\Phi_e^{D(\mathcal{B}')}$ es un isomorfismo de $\mathcal{B}'$ a $\mathcal{A}$. Como g' y $\Phi_e^{D(\mathcal{B}')}$ son isomorfismos de $\mathcal{B}'$ a $\mathcal{A}$ entonces tenemos que

$$(\mathcal{A}, g \upharpoonright \overline{n}) \cong (\mathcal{B}', \overline{n}) \cong (\mathcal{A}, \Phi_e^{D(\mathcal{B}')} \upharpoonright \overline{n})$$

Como $\overline{q} \subseteq \overline{g'}$, tenemos que $D_{\mathcal{A}}(\overline{q}) \subseteq D(\mathcal{B}')$ y $\overline{g} \upharpoonright \overline{n} = \overline{q} \upharpoonright \overline{n}$. Dado que $\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright \overline{n} \downarrow$ entonces $\Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright \overline{n} = \Phi_e^{D(\mathcal{B}')} \upharpoonright \overline{n}$. Por consiguiente

$$(\mathcal{A}, \overline{q} \upharpoonright \overline{n}) \cong (\mathcal{B}', \overline{n}) \cong (\mathcal{A}, \Phi_e^{D_{\mathcal{A}}(\overline{q})} \upharpoonright \overline{n})$$

Para $\overline{r}$, sea g^* su extensión y $\mathcal{B}^* = (g^*)^{-1}(\mathcal{A})$. Entonces $\Phi_e^{D(\mathcal{B}^*)}$ es un isomorfismo de $\mathcal{B}^*$ a $\mathcal{A}$. De manera similar a la anterior podemos concluir que

$$(\mathcal{A}, \overline{r} \upharpoonright \overline{n}) \cong (\mathcal{B}^*, \overline{n}) \cong (\mathcal{A}, \Phi_e^{D_{\mathcal{A}}(\overline{r})} \upharpoonright \overline{n})$$

De lo mencionado anteriormente concluimos que

$$\begin{aligned} (\mathcal{A}, \overline{q} \upharpoonright \overline{n}) &\cong (\mathcal{A}, \Phi_e^{D_{\mathcal{A}}(\overline{q}|\overline{n})} \upharpoonright \overline{n}) \\ &= (\mathcal{A}, \Phi_e^{D_{\mathcal{A}}(\overline{r}|\overline{n})} \upharpoonright \overline{n}) \\ &\cong (\mathcal{A}, \overline{r} \upharpoonright \overline{n}) \end{aligned}$$

Lo cual demuestra la afirmación mencionada previamente.

Vayamos al último paso de esta demostración construyendo la familia de Scott computable enumerable de fórmulas existenciales para la estructura $(\mathcal{A}, \overline{p})$. Para ello sea $\overline{a} \in \mathcal{A}$. Por lo tanto podemos obtener de forma computable un par de tuplas $\overline{q}_{\overline{a}}, \overline{n}_{\overline{a}} \in A^{<\mathbb{N}}$ tal que $len(\overline{n}_{\overline{a}}) \leq len(\overline{q}_{\overline{a}})$, todo elemento de $\overline{n}_{\overline{a}}$ es menor a $len(\overline{q}_a)$, $\overline{q}_{\overline{a}} \supseteq \overline{p}$, $\overline{q}_{\overline{a}} \upharpoonright \overline{n}_{\overline{a}} = \overline{pa}$ y $\Phi_e^{D_{\mathcal{A}}(\overline{q}_{\overline{a}})} \upharpoonright \overline{n}_{\overline{a}} \downarrow$.

En efecto primero agregamos a $\overline{p}$ los elementos de $\overline{a}$ y consideramos $\overline{n}_a$ como la tupla de las posiciones de la tupla $\overline{pa}$. La longitud de $\overline{n}_a$ es la longitud de $\overline{pa}$ y todos sus elementos son menores a la longitud de $\overline{pa}$. Ahora notemos que $\Phi_e^{D(\mathcal{B})}(n)\downarrow$ en cada elemento n de la tupla $\overline{n}_{\overline{a}}$. Como $\Phi_e^{D(\mathcal{B})}$ es una función total debemos obtener que dicho programas se detienen. Sin embargo al final de dichas ejecuciones observaremos que se han realizado una cantidad finita de consultas al oráculo $D(\mathcal{B})$ por cada elemento de $\overline{n}_a$. Más aún como $D(\mathcal{B})$ es la unión de los diagramas de la forma $D_{\mathcal{A}}(g \upharpoonright k)$ para cada $k \in \omega$, entonces debe existir $k_0 \in \omega$ tal que las consultas al oráculo $D_{\mathcal{A}}(g \upharpoonright k_0)$ son suficientes para tener que $\Phi_e^{D_{\mathcal{A}}(g\upharpoonright k_0)} \upharpoonright \overline{n}_{\overline{a}}\downarrow$. Por lo tanto basta agregar a $\overline{pa}$ elementos suficientes que no hayan sido agregados hasta ese momento y comprobar de forma simultánea si Φ_e con oráculo el diagrama en $\mathcal{A}$ de la tupla obtenida se detiene en todos los elementos de $\overline{n}_{\overline{a}}$. Este proceso de verificación simultánea debe terminar ya que en algun momento habremos agregado a $\overline{pa}$ los elementos de $g \upharpoonright k_0$. Cuando ocurra esto el programa se detiene devolviendo la tupla obtenida, que nombraremos $\overline{q}_{\overline{a}}$.

Afirmamos que para cualquier tupla $\overline{b} \in A^{<\mathbb{N}}$ tenemos que

$$(\mathcal{A}, \overline{pa}) \cong (\mathcal{A}, \overline{p}\overline{b}) \text{ si y sólo si } \exists\, \overline{q}(\overline{q} \supseteq \overline{p} \ \wedge\ \overline{q} \upharpoonright \overline{n}_{\overline{a}} = \overline{p}\overline{b} \ \wedge\ D_{\mathcal{A}}(\overline{q}) = D_{\mathcal{A}}(\overline{q}_{\overline{a}}))$$

Supongamos que $(\mathcal{A}, \overline{pa}) \cong (\mathcal{A}, \overline{p}\overline{b})$. Sea h un automorfismo de $\mathcal{A}$ tal que h manda a $\overline{pa}$ en $\overline{p}\overline{b}$. Sea $\overline{q}$ la tupla obtenida por h en $\overline{q}_{\overline{a}}$. Entonces se debe satisfacer que $D_{\mathcal{A}}(\overline{q}) = D_{\mathcal{A}}(\overline{q}_{\overline{a}})$ por la definición de $\overline{q}$ y por la condición de preservación de fórmulas atómicas de h. Como $\overline{q}_{\overline{a}} \upharpoonright \overline{n}_{\overline{a}} = \overline{pa}$ entonces como h manda a $\overline{pa}$ a $\overline{p}\overline{b}$ y $\overline{q}_{\overline{a}}$ a $\overline{q}$ entonces tenemos que $\overline{q} \upharpoonright \overline{n}_{\overline{a}} = \overline{p}\overline{b}$. Por lo tanto $\overline{q} \supseteq \overline{p}$. Con lo cual queda demostrada la formula de la parte derecha.

Ahora supongamos dicha fórmula y veamos que $(\mathcal{A}, \overline{pa}) \cong (\mathcal{A}, \overline{p}\overline{b})$. Sea $\overline{q}$ una tupla dada por la fórmula ya mencionada. Notemos que tanto $\overline{q}$ como $\overline{q}_{\overline{a}}$ satisfacen las condiciones de la segunda afirmación de este teorema. Como $D_{\mathcal{A}}(\overline{q}_{\overline{a}}) = D_{\mathcal{A}}(\overline{q})$ entonces tenemos que $(\mathcal{A}, \overline{q}_{\overline{a}} \upharpoonright \overline{n}_{\overline{a}}) \cong (\mathcal{A}, \overline{q} \upharpoonright \overline{n}_{\overline{a}})$, lo cual implica que $(\mathcal{A}, \overline{pa}) \cong (\mathcal{A}, \overline{p}\overline{b})$. Con esto último queda demostrada la equivalencia anterior.

Definimos la fórmula existencial $\varphi_{\overline{a}}(\overline{p}, \overline{x})$ con parámetros $\overline{p}$ como la fórmula

$$\exists \overline{y}(\overline{y} \supseteq \overline{p} \ \wedge\ \overline{y} \upharpoonright \overline{n}_{\overline{a}} = \overline{px} \ \wedge\ D_{\mathcal{A}}(\overline{y}) = D_{\mathcal{A}}(\overline{q}_{\overline{a}}))$$

Por el momento no es claro que la fórmula anterior se pueda ver estrictamente como una fórmula existencial por la parte $\exists \overline{y}$, la cual a primera vista nos dice que deberíamos fijarnos en una tupla finita de una longitud específica. Afortunadamente podemos codificar tuplas finitas de tamaño arbitrario por un número natural. Por lo tanto basta considerar la fórmula anterior como una fórmula de la siguiente forma:

$$\exists n((\overline{n})_0 = p_0 \ \wedge \ \cdots \ \wedge \ (\overline{n})_{len(\overline{p})-1} = p_{len(\overline{p})-1} \ \wedge \ (\overline{n})_{((\overline{n}_{\overline{a}})_{len(\overline{p})})} = x_0 \ \wedge \ \cdots \wedge$$
$$\wedge \ (\overline{n})_{((\overline{n}_{\overline{a}})_{len(\overline{n}_{\overline{a}})-1)} = x_{len(\overline{n}_{\overline{a}})-1-len(\overline{p})} \ \wedge \ D_{\mathcal{A}}(\overline{n}) \subseteq D_{\mathcal{A}}(\overline{q}_{\overline{a}}) \ \wedge \ D_{\mathcal{A}}(\overline{q}_{\overline{a}}) \subseteq D_{\mathcal{A}}(\overline{n}))$$

donde denotamos por $\overline{n}$ a la tupla finita asociada a n bajo la función $\boldsymbol{\psi}$, $(\overline{n})_k$ es el k-ésimo elemento de la tupla $\overline{n}$ y las fórmulas $D_{\mathcal{A}}(\overline{y}) \subseteq D_{\mathcal{A}}(\overline{q}_{\overline{a}})$ y $D_{\mathcal{A}}(\overline{q}_{\overline{a}}) \subseteq D_{\mathcal{A}}(\overline{n})$ podemos verlas como fórmulas libres de cuantificadores dadas por la observación 4,2,1.

En observaciones anteriores obtuvimos a $\overline{q}_{\overline{a}}$ y a $\overline{n}_{\overline{a}}$ de forma computable a partir de $\overline{a}$. Como $\mathcal{A}$ es computable, podemos obtener a $D_{\mathcal{A}}(\overline{q}_{\overline{a}})$ de forma computable. Esto implica que podemos obtener de forma computable a las fórmulas libres de cuantificadores mencionadas anteriormente. Los demás componentes de la fórmula $\varphi_{\overline{a}}(\overline{p}, \overline{x})$ los podemos obtener a partir de la longitud de $\overline{a}$ y a partir de $\overline{n}_{\overline{a}}$. Por lo tanto dado $\overline{a}$ podemos obtener de forma computable su fórmula $\varphi_{\overline{a}}(\overline{p}, \overline{x})$. Esto hace que el conjunto de fórmulas existenciales con parámetros en $\overline{p}$, $\{\varphi_{\overline{a}} : \overline{a} \in A^{<\mathbb{N}}\}$ sea computable enumerable.

El paso que nos falta es ver que en efecto $\{\varphi_{\overline{a}} : \overline{a} \in A^{<\mathbb{N}}\}$ es una familia de Scott. Para ello notemos que $\overline{a}$ satisface su fórmula $\varphi_{\overline{a}}$ en $\mathcal{A}$ ya que $\overline{q}_{\overline{a}} \supseteq \overline{p}$, $\overline{q}_{\overline{a}} \restriction \overline{n}_{\overline{a}} = \overline{pa}$ y $D_{\mathcal{A}}(\overline{q}_{\overline{a}}) = D_{\mathcal{A}}(\overline{q}_{\overline{a}})$. Por otro lado si $\overline{b}$ es una tupla finita que satisface $\varphi_{\overline{a}}$ en $\mathcal{A}$ entonces definimos como $\overline{q}$ a la tupla testigo de tal hecho. Por una observación anterior tenemos que $(\mathcal{A}, \overline{pa}) \cong (\mathcal{A}, \overline{p}\overline{b})$, es decir existe un automorfismo de $(\mathcal{A}, \overline{p})$ que manda a $\overline{a}$ a $\overline{b}$. Por lo tanto $\{\varphi_{\overline{a}} : \overline{a} \in A^{<\mathbb{N}}\}$ es una familia de Scott.

Concluimos que $\{\varphi_{\overline{a}} : \overline{a} \in A^{<\mathbb{N}}\}$ es una familia de Scott computable enumerable de fórmulas existenciales para la estructura $(\mathcal{A}, \overline{p})$, lo cual demuestra el inciso *(iii)* y termina con la demostración de este teorema.

□

Apéndice

Programa de minimización

En la proposición 1,2,2 dimos el ejemplo de un programa escrito en pseudocódigo, es decir escrita con abreviaciones de instrucciones para que fuera más entendible. A continuación mostraremos su programa equivalente en el lenguaje de las máquinas de acceso aleatorio y lo comparamos con el programa escrito en dicha proposición. Sea l el número de instrucciones de $(\mathbf{F}^{[m]})^{(m)}$. Sea $j = 3 + 1 + n + 1 + l + 3 + 2$.

Programa en pseudocódigo	**Programa en instrucciones RAM**
$r_{n+1} \leftarrow 0$	$r_{n+1} \leftarrow 0$
$r_{n+2} \leftarrow 0$	$r_{n+2} \leftarrow 0$
$r_{n+3} \leftarrow r_{n+3} + 1$	$r_{n+3} \leftarrow r_{n+3} + 1$
	si $r_{n+3} = r_{n+2}$ **entonces**
	ir a instrucción número $j - 1$
	en otro caso
	ir a instrucción siguiente
hacer	
$r_{m+1} \leftarrow r_1$	$r_{m+1} \leftarrow r_1$
$\vdots$	$\vdots$
$r_{m+n+1} \leftarrow r_{n+1}$	$r_{m+n+1} \leftarrow r_{n+1}$
$(\mathbf{F}^{[m]})^{(m)}$	$(\mathbf{F}^{[m]})^{(m)}$
$r_{n+3} \leftarrow r_{m+1}$	$r_{n+3} \leftarrow r_{m+1}$
$r_{n+1} \leftarrow r_{n+1} + 1$	$r_{n+1} \leftarrow r_{n+1} + 1$
	si $r_1 = r_1$ **entonces**
	ir a instrucción número 4
	en otro caso
	ir a instrucción siguiente
mientras $r_{n+3} \neq 0$	
$r_{n+1} \leftarrow r_{n+1} \dot{-} 1$	$r_{n+1} \leftarrow r_{n+1} \dot{-} 1$
$r_1 \leftarrow r_{n+1}$	$r_1 \leftarrow r_{n+1}$

La comparación anterior nos hace más explícito que la cantidad de instrucciones del programa $\mathbf{H}$ es j. Además que la instrucción $j-2$ siempre hace que la instrucción 4 se ejecute. Por lo tanto en el tiempo de ejecución $r = 3 + \sum_{j=0}^{y} (1 + (n+1) + t_j + 3) + 4$

se justifica el último 4 ya que se ejecuta una vez la instrucción 4, se ejecutan las instrucciones $j-1$ y j y se ejecuta la instrucción que da salida al programa. Además la ejecución simulada en dicha proposición ya tomamos en cuenta al programa en sus instrucciones RAM.

Programa de recursión primitiva

En la proposición 1,2,3 dimos el ejemplo de un programa escrito en pseudocódigo para que fuera más entendible. A continuación demostraremos que dicho programa tiene su equivalente escrito sólo con instrucciones RAM. Sea $j = j = 1 + n + l_1 + 2 + n + l_2 + 3 + 1$. Entonces,

Programa en pseudocódigo	**Programa en instrucciones RAM**
$r_{n+2} \leftarrow 0$	$r_{n+2} \leftarrow 0$
$r_{q+1} \leftarrow r_1$	$r_{q+1} \leftarrow r_1$
$\vdots$	$\vdots$
$r_{q+n} \leftarrow r_n$	$r_{q+n} \leftarrow r_n$
$(\mathbf{F}^{[q]})^{(q)}$	$(\mathbf{F}^{[q]})^{(q)}$
$r_{n+3} \leftarrow r_{q+1}$	$r_{n+3} \leftarrow r_{q+1}$
	si $r_{n+1} = r_{n+2}$ **entonces**
	ir a instrucción número j
	en otro caso
	ir a instrucción siguiente
mientras $r_{n+1} \neq r_{n+2}$	$r_{q+1} \leftarrow r_1$
$r_{q+1} \leftarrow r_1$	$\vdots$
$\vdots$	$r_{q+n} \leftarrow r_n$
$r_{q+n} \leftarrow r_n$	$r_{q+n+1} \leftarrow r_{n+2}$
$r_{q+n+1} \leftarrow r_{n+2}$	$r_{q+n+2} \leftarrow r_{n+3}$
$r_{q+n+2} \leftarrow r_{n+3}$	$(\mathbf{G}^{[q]})^{(q)}$
$(\mathbf{G}^{[q]})^{(q)}$	$r_{n+3} \leftarrow r_{q+1}$
$r_{n+3} \leftarrow r_{q+1}$	$r_{n+2} \leftarrow r_{n+2} + 1$
$r_{n+2} \leftarrow r_{n+2} + 1$	**si** $r_1 = r_1$ **entonces**
	ir a instrucción número $1 + n + l_1 + 2$
fin mientras	**en otro caso**
	ir a instrucción siguiente
$r_1 \leftarrow r_{n+3}$	$r_1 \leftarrow r_{n+3}$

La comparación anterior hace explícito que en efecto j es la cantidad de instrucciones del programa anterior.

Detalles de la sección 1.3

En este apartado nos interesa dar demostraciones más formales de las proposiciones 1.3.1 y 1.3.2.

Primero mostramos un programa que nos ayudará a trasladar todo un bloque de registros a registros más adelante de la cinta de trabajo, de modo que si tenemos una instrucción de carga o de almacenamiento que se refiera a direcciones de registro que involucran índices suficientemente grandes, tengamos la certeza de que no hay problema con interrumpir el sentido de un programa si guardamos elementos después de su espacio de trabajo.

Por ello damos un programa que nos servirá con este propósito y del cual pediremos que se encuentre antes de las instrucciones de carga o de almacenamiento.

Sea **Q** el siguiente programa.

Programa en pseudocódigo	**Explicación del programa**
si $r_n > r_1$ **entonces**	Si el índice de la instrucción de carga o almacenamiento es mayor al índice que indica el final del espacio de trabajo del programa entonces se ejecuta lo siguiente.
$r_2 \leftarrow r_{r_1+1}$	Asignamos a r_2 el contenido de r_{r_1+1} que contiene la cantidad total de elementos después de los dos registros siguientes del espacio de trabajo.
mientras $r_2 > 0$	Iniciamos un ciclo en el que se realizará una transferencia de los elementos después del espacio de trabajo inicial a un espacio de trabajo después de la dirección marcada por r_n.
$r_{r_n+2+r_2} \leftarrow r_{r_1+2+r_2}$	Movemos el elemento mencionado. Comenzamos a mover dichos elementos de derecha a izquierda para que no existan problema alguno con la eliminación del contenido de un registro.

$r_2 \leftarrow r_2 \dot{-} 1$	Disminuimos en uno el contador r_2 para poder terminar el ciclo.
fin mientras	Finalizamos el ciclo cuando se cumple que $r_2 = 0$
$r_{r_n+2} \leftarrow r_{r_1+2}$	El valor del segundo registro posterior al espacio de trabajo lo movemos al registro posterior al nuevo espacio de trabajo.
$r_{r_n+1} \leftarrow r_{r_1+1}$	Lo mismo que en la instrucción anterior pero con el registro inemdiato posterior al espacio de trabajo.
mientras $r_1 < r_n$	Iniciaremos un ciclo en el que daremos el valor 0 a todos los registros que se encuentren en medio del último registro del espacio de trabajo original y el primer registro posterior al nuevo espacio de trabajo.
$r_1 \leftarrow r_1 + 1$	Aumentamos en uno el valor de r_1 a fin de que alcance el valor de r_n que será su nuevo valor.
$r_{r_1} \leftarrow 0$	Asignamos 0 a dicho registro.
fin mientras	Finalizamos el ciclo. El valor de r_1 ahora es r_n.
fin si	Finalizamos el condicional.

A continuación daremos una descripción del programa anterior en términos de instrucciones RAM.

Programa en pseudocódigo	**Programa en instrucciones RAM**
	$\mathbf{I}_1:$ $r_2 \leftarrow r_n$
	$\mathbf{I}_2:$ $r_3 \leftarrow 0$
	$\mathbf{I}_3:$ $r_4 \leftarrow 0$
	$\mathbf{I}_4:$ $r_5 \leftarrow 0$
	$\mathbf{I}_5:$ $r_6 \leftarrow 0$
	$\mathbf{I}_6:$ $r_7 \leftarrow 0$
	$\mathbf{I}_7:$ **si** $r_3 = r_1$ **entonces**
	ir a instrucción número 11
	en otro caso
	ir a instrucción siguiente
	$\mathbf{I}_8:$ $r_2 \leftarrow r_2 \dot{-} 1$
	$\mathbf{I}_9:$ $r_3 \leftarrow r_3 + 1$
si $r_n > r_1$ **entonces**	$\mathbf{I}_{10}:$ **si** $r_1 = r_1$ **entonces**
	ir a instrucción número 7
	en otro caso
	ir a instrucción siguiente
	$\mathbf{I}_{11}:$ $r_3 \leftarrow 0$
	$\mathbf{I}_{12}:$ **si** $r_2 = r_3$ **entonces**
	ir a instrucción número 14
	en otro caso
	ir a instrucción siguiente
	$\mathbf{I}_{13}:$ **si** $r_1 = r_1$ **entonces**
	ir a instrucción número 44
	en otro caso
	ir a instrucción siguiente
	$\mathbf{I}_{14}:$ $r_3 \leftarrow r_1$
$r_2 \leftarrow r_{r_1+1}$	$\mathbf{I}_{15}:$ $r_3 \leftarrow r_3 + 1$
	$\mathbf{I}_{16}:$ $r_2 \leftarrow r_{r_3}$
	$\mathbf{I}_{17}:$ $r_3 \leftarrow r_3 + 1$
	$\mathbf{I}_{18}:$ $r_4 \leftarrow 0$
	$\mathbf{I}_{19}:$ **si** $r_2 = r_4$ **entonces**
mientras $r_2 > 0$	**ir a** instrucción número 30
	en otro caso
	ir a instrucción siguiente

	$\mathbf{I}_{20}:\ r_6 \leftarrow r_n$
	$\mathbf{I}_{21}:\ r_6 \leftarrow r_6 + 1$
	$\mathbf{I}_{22}:\ r_6 \leftarrow r_6 + 1$
	$\mathbf{I}_{23}:\ r_6 \leftarrow r_6 + r_2$
$r_{r_n+2+r_2} \leftarrow r_{r_1+2+r_2}$	$\mathbf{I}_{24}:\ r_5 \leftarrow r_3$
	$\mathbf{I}_{25}:\ r_5 \leftarrow r_5 + r_2$
	$\mathbf{I}_{26}:\ r_7 \leftarrow r_{r_5}$
	$\mathbf{I}_{27}:\ r_{r_6} \leftarrow r_7$
$r_2 \leftarrow r_2 \dot{-} 1$	$\mathbf{I}_{28}:\ r_2 \leftarrow r_2 \dot{-} 1$
	$\mathbf{I}_{29}:$ **si** $r_1 = r_1$ **entonces**
fin mientras	**ir a** instrucción número 19
	en otro caso
	ir a instrucción siguiente
	$\mathbf{I}_{30}:\ r_4 \leftarrow r_n$
	$\mathbf{I}_{31}:\ r_4 \leftarrow r_4 + 1$
$r_{r_n+2} \leftarrow r_{r_1+2}$	$\mathbf{I}_{32}:\ r_4 \leftarrow r_4 + 1$
	$\mathbf{I}_{33}:\ r_5 \leftarrow r_{r_3}$
	$\mathbf{I}_{34}:\ r_{r_4} \leftarrow r_5$
	$\mathbf{I}_{35}:\ r_3 \leftarrow r_3 \dot{-} 1$
	$\mathbf{I}_{36}:\ r_4 \leftarrow r_4 \dot{-} 1$
$r_{r_n+1} \leftarrow r_{r_1+1}$	$\mathbf{I}_{37}:\ r_5 \leftarrow r_{r_3}$
	$\mathbf{I}_{38}:\ r_{r_4} \leftarrow r_5$
	$\mathbf{I}_{39}:$ **si** $r_1 = r_n$ **entonces**
mientras $r_1 < r_n$	**ir a** instrucción número 44
	en otro caso
	ir a instrucción siguiente
$r_1 \leftarrow r_1 + 1$	$\mathbf{I}_{40}:\ r_1 \leftarrow r_1 + 1$
	$\mathbf{I}_{41}:\ r_2 \leftarrow 0$
$r_{r_1} \leftarrow 0$	$\mathbf{I}_{42}:\ r_{r_1} \leftarrow r_2$
	$\mathbf{I}_{43}:$ **si** $r_1 = r_1$ **entonces**
fin mientras	**ir a** instrucción número 39
	en otro caso
	ir a instrucción siguiente

fin si

Dado un programa $\mathbf{P}$, definimos como $\mathbf{P}^*$ al programa que resulta de añadir el programa $\mathbf{Q}$ inmediatamente antes de cada instrucción de carga o almacenamiento de $\mathbf{P}$, en caso de que $\mathbf{P}$ tenga instrucciones de este tipo.

Detalles de la proposición 1.3.1

A continuación daremos con mayor precisión un programa que hace a la función configuración de la definición 1,3,1 una función calculable.

Como mencionamos en la demostración de la proposición 1,3,1, la idea del programa, es que a partir de la cantidad de elementos de la tupla finita dada podamos saber cuántos números primos vamos a calcular, que posteriormente vamos a calcular sus potencias con respecto a los números de la tupla dada y finalmente vamos a obtener el producto de todos ellos.

Para ello sean $\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{Q}_3$ los programas de las funciones p, exponente y producto entre dos números naturales respectivamente. [1]

Sea $m = \text{máx}\{\rho_{\mathbf{Q}_1}, \rho_{\mathbf{Q}_2}, \rho_{\mathbf{Q}_3}\} + 11$. Definimos como $\mathbf{P}$ al siguiente programa:

Programa en pseudocódigo	**Explicación del programa**
$r_{m-1} \leftarrow r_m$	Asignamos la cantidad de elementos de la tupla inicial al registro que contiene la cantidad de elementos que existen después de los dos primeros registros posteriores al espacio de trabajo inicial.
$r_1 \leftarrow 0$	Nos aseguramos de que el registro r_1 tenga el valor de 0 para poder asignarle el valor de la posición final del espacio de trabajo.
$\left.\begin{array}{l} r_1 \leftarrow r_1 + 1 \\ \quad\vdots \\ r_1 \leftarrow r_1 + 1 \end{array}\right\} m \dot{-} 2$	Asignamos $m \dot{-} 2$ al registro r_1.
$r_8 \leftarrow r_m$	Damos el valor de r_m a r_8, que es la cantidad de elementos iniciales para ser utilizados como contador.

[1] Para recordar cuáles on esta funciones vaya al lema 1,2,1 y al ejemplo 1,2,1

mientras $r_8 > 0$	Iniciamos un ciclo en el que vamos a obtener los primeros r_m números primos y guardaremos después de la entrada inicial.
$r_{10} \leftarrow r_8$	Colocamos el contenido del registro r_8 como entrada del programa que calcula el r_8-número primo.
$((\mathbf{Q}_1^{[9]})^{(9)})^*$	Calculamos dicho número.
$r_{r_1+2+r_{r_1+2}+r_8} \leftarrow r_{10}$	Guardamos el número primo obtenido en una posición posterior a la entrada inicial. Este guardado será de derecha a izquierda para ir con el sentido del contador r_8.
$r_{r_1+1} \leftarrow r_{r_1+1} + 1$	Aumentamos en uno el número de elementos después de los dos primeros registros posteriores a la posición inicial.
$r_8 \leftarrow r_8 \dot{-} 1$	Disminuimos en uno el contador.
fin mientras	Finalizamos el ciclo cuando el contador sea cero. En este paso se habrán calculado los números primos necesarios.
$r_8 \leftarrow r_{r_1+2}$	Reiniciamos el contador r_8 con la cantidad de elementos en la tupla dada inicialmente.
mientras $r_8 > 0$	Iniciamos un nuevo ciclo en el que calculamos los resultados de elevar un número primo al elemento de la entrada más uno que le corresponda.
$r_{11} \leftarrow r_{r_1+2+r_8}$	Colocamos como entrada al elemento correspondiente de la entrada original.
$r_{10} \leftarrow r_{r_1+2+r_{r_1+2}+r_8}$	Colocamos como entrada al número primo correspondiente que hayamos obtenido en el ciclo anterior.

$r_{11} \leftarrow r_{11} + 1$	Sumamos uno al elemento que tomaremos como potencia
$((\mathbf{Q}_2^{[9]})^{(9)})^*$	Ejecutamos el programa que calcula el valor de la potencia de r_{10} elevado al contenido r_{11}.
$r_{r_1+2+r_{r_1+2}+r_{r_1+2}+r_8} \leftarrow r_{10}$	El resultado obtenido lo guardamos en registros posteriores a los registros donde se guardaron los números primos.
$r_{r_1+1} \leftarrow r_{r_1+1} + 1$	Aumentamos en uno la cantidad de elementos que hay después de los dos primeros registros posteriores al espacio de trabajo actual.
$r_8 \leftarrow r_8 \dot{-} 1$	Disminuimos en uno el contador.
fin mientras	Finalizamos el ciclo cuando este proceso lo hayamos repetido con cada elemento de la tupla dada inicialmente con su respectivo número primo.
$r_8 \leftarrow r_{r_1+2}$	Nuevamente asignamos al contador r_8 la cantidad de elementos que hay en la tupla dada inicialmente.
$r_{10} \leftarrow r_{r_1+2+r_{r_1+2}+r_{r_1+2}+r_8}$	Tomamos la potencia que se encuentra más a la derecha, o bien la primera potencia que calculamos en el ciclo anterior y la colocamos como entrada del programa que calcula el producto de dos elementos.
$r_8 \leftarrow r_8 \dot{-} 1$	Disminuimos en uno el contador.
mientras $r_8 > 0$	Iniciaremos un ciclo en el que calcularemos el producto entre sí de todas las potencias obtenidas en el ciclo anterior.

$r_{11} \leftarrow r_{r_1+2+r_{r_1+2}+r_{r_1+2}+r_8}$	Colocamos como entrada del programa que calcula el producto de dos números a la potencia inmediata a la izquierda de la potencia tomada anteriormente.
$((\mathbf{Q}_3^{[9]})^{(9)})^*$	Calculamos el producto de el producto obtenido hasta el momento y la potencia colocada recientemente.
$r_8 \leftarrow r_8 \dot{-} 1$	Disminuimos en uno el contador.
fin mientras	Finalizamos el ciclo cuando hayamos calculado el producto de las potencias.
$r_1 \leftarrow r_{10}$	El resultado final que se encuentra en el registro r_{10} lo colocamos en el registro r_1 para que sea devuelto por el programa.

A continuación veremos que el programa anterior es en efecto un programa en instrucciones RAM.

Programa en pseudocódigo	**Programa en instrucciones RAM**
$r_{m-1} \leftarrow r_m$	$\mathbf{I}_1:\ r_{m-1} \leftarrow r_m$
$r_1 \leftarrow 0$	$\mathbf{I}_2:\ r_1 \leftarrow 0$
$\left.\begin{array}{l} r_1 \leftarrow r_1 + 1 \\ \quad\vdots \\ r_1 \leftarrow r_1 + 1 \end{array}\right\} m \dot{-} 2$	$\left.\begin{array}{l} \mathbf{I}_3:\ r_1 \leftarrow r_1 + 1 \\ \quad\vdots \\ \mathbf{I}_m:\ r_1 \leftarrow r_1 + 1 \end{array}\right\} m \dot{-} 2$
$r_8 \leftarrow r_m$	$\mathbf{I}_{m+1}:\ r_8 \leftarrow r_m$
mientras $r_8 > 0$	$\mathbf{I}_{m+2}:\ r_9 \leftarrow 0$ $\mathbf{I}_{m+3}:$ **si** $r_8 = r_9$ **entonces** **ir a** instrucción número $m + 4 + l_1 + 15$ **en otro caso** **ir a** instrucción siguiente
$r_{10} \leftarrow r_8$	$\mathbf{I}_{m+4}:\ r_{10} \leftarrow r_8$
$((\mathbf{Q}_1^{[9]})^{(9)})^*$	$((\mathbf{Q}_1^{[9]})^{(9)})^*$

	$\mathbf{I}_{m+4+l_1+1}:\ r_2 \leftarrow r_1$
	$\mathbf{I}_{m+4+l_1+2}:\ r_2 \leftarrow r_2 + 1$
	$\mathbf{I}_{m+4+l_1+3}:\ r_2 \leftarrow r_2 + 1$
$r_{r_1+2+r_{r_1+2}+r_8} \leftarrow r_{10}$	$\mathbf{I}_{m+4+l_1+4}:\ r_3 \leftarrow r_{r_2}$
	$\mathbf{I}_{m+4+l_1+5}:\ r_2 \leftarrow r_2 + r_3$
	$\mathbf{I}_{m+4+l_1+6}:\ r_2 \leftarrow r_2 + r_8$
	$\mathbf{I}_{m+4+l_1+7}:\ r_{r_2} \leftarrow r_{10}$
	$\mathbf{I}_{m+4+l_1+8}:\ r_2 \leftarrow r_1$
	$\mathbf{I}_{m+4+l_1+9}:\ r_2 \leftarrow r_2 + 1$
$r_{r_1+1} \leftarrow r_{r_1+1} + 1$	$\mathbf{I}_{m+4+l_1+10}:\ r_3 \leftarrow r_{r_2}$
	$\mathbf{I}_{m+4+l_1+11}:\ r_3 \leftarrow r_3 + 1$
	$\mathbf{I}_{m+4+l_1+12}:\ r_{r_2} \leftarrow r_3$
$r_8 \leftarrow r_8 \dot{-} 1$	$\mathbf{I}_{m+4+l_1+13}:\ r_8 \leftarrow r_8 \dot{-} 1$
	$\mathbf{I}_{m+4+l_1+14}:$ **si** $r_1 = r_1$ **entonces**
	ir a instrucción número
fin mientras	$m+3$
	en otro caso
	ir a instrucción siguiente
	$\mathbf{I}_{m+4+l_1+15}:\ r_2 \leftarrow r_1$
	$\mathbf{I}_{m+4+l_1+16}:\ r_2 \leftarrow r_2 + 1$
$r_8 \leftarrow r_{r_1+2}$	$\mathbf{I}_{m+4+l_1+17}:\ r_2 \leftarrow r_2 + 1$
	$\mathbf{I}_{m+4+l_1+18}:\ r_8 \leftarrow r_{r_2}$
	$\mathbf{I}_{m+4+l_1+19}:\ r_9 \leftarrow 0$
	$\mathbf{I}_{m+4+l_1+20}:$ **si** $r_8 = r_9$ **entonces**
	ir a instrucción número
mientras $r_8 > 0$	$m+4+l_1+33+l_2+16$
	en otro caso
	ir a instrucción siguiente
	$\mathbf{I}_{m+4+l_1+21}:\ r_2 \leftarrow r_1$
	$\mathbf{I}_{m+4+l_1+22}:\ r_2 \leftarrow r_2 + 1$
$r_{11} \leftarrow r_{r_1+2+r_8}$	$\mathbf{I}_{m+4+l_1+23}:\ r_2 \leftarrow r_2 + 1$
	$\mathbf{I}_{m+4+l_1+24}:\ r_2 \leftarrow r_2 + r_8$
	$\mathbf{I}_{m+4+l_1+25}:\ r_{11} \leftarrow r_{r_2}$

	$\mathbf{I}_{m+4+l_1+26}:\ r_2 \leftarrow r_1$ $\mathbf{I}_{m+4+l_1+27}:\ r_2 \leftarrow r_2+1$ $\mathbf{I}_{m+4+l_1+28}:\ r_2 \leftarrow r_2+1$
$r_{10} \leftarrow r_{r_1+2+r_{r_1+2}+r_8}$	$\mathbf{I}_{m+4+l_1+29}:\ r_3 \leftarrow r_{r_2}$ $\mathbf{I}_{m+4+l_1+30}:\ r_2 \leftarrow r_2+r_3$ $\mathbf{I}_{m+4+l_1+31}:\ r_2 \leftarrow r_2+r_8$ $\mathbf{I}_{m+4+l_1+32}:\ r_{10} \leftarrow r_{r_2}$
$r_{11} \leftarrow r_{11}+1$	$\mathbf{I}_{m+4+l_1+33}:\ r_{11} \leftarrow r_{11}+1$
$((\mathbf{Q}_2^{[9]})^{(9)})^*$	$((\mathbf{Q}_2^{[9]})^{(9)})^*$
$r_{r_1+2+r_{r_1+2}+r_{r_1+2}+r_8} \leftarrow r_{10}$	$\mathbf{I}_{m+4+l_1+33+l_2+1}:\ r_2 \leftarrow r_1$ $\mathbf{I}_{m+4+l_1+33+l_2+2}:\ r_2 \leftarrow r_2+1$ $\mathbf{I}_{m+4+l_1+33+l_2+3}:\ r_2 \leftarrow r_2+1$ $\mathbf{I}_{m+4+l_1+33+l_2+4}:\ r_3 \leftarrow r_{r_2}$ $\mathbf{I}_{m+4+l_1+33+l_2+5}:\ r_2 \leftarrow r_2+r_3$ $\mathbf{I}_{m+4+l_1+33+l_2+6}:\ r_2 \leftarrow r_2+r_3$ $\mathbf{I}_{m+4+l_1+33+l_2+7}:\ r_2 \leftarrow r_2+r_8$ $\mathbf{I}_{m+4+l_1+33+l_2+8}:\ r_{r_2} \leftarrow r_{10}$
$r_{r_1+1} \leftarrow r_{r_1+1}+1$	$\mathbf{I}_{m+4+l_1+33+l_2+9}:\ r_2 \leftarrow r_1$ $\mathbf{I}_{m+4+l_1+33+l_2+10}:\ r_2 \leftarrow r_2+1$ $\mathbf{I}_{m+4+l_1+33+l_2+11}:\ r_3 \leftarrow r_{r_2}$ $\mathbf{I}_{m+4+l_1+33+l_2+12}:\ r_3 \leftarrow r_3+1$ $\mathbf{I}_{m+4+l_1+33+l_2+13}:\ r_{r_2} \leftarrow r_3$
$r_8 \leftarrow r_8 \dot{-} 1$	$\mathbf{I}_{m+4+l_1+33+l_2+14}:\ r_8 \leftarrow r_8 \dot{-} 1$
fin mientras	$\mathbf{I}_{m+4+l_1+33+l_2+15}:$ **si** $r_1 = r_1$ **entonces** **ir a** instrucción $m+4+l_1+20$ **en otro caso** **ir a** instrucción siguiente
$r_8 \leftarrow r_{r_1+2}$	$\mathbf{I}_{m+4+l_1+33+l_2+16}:\ r_2 \leftarrow r_1$ $\mathbf{I}_{m+4+l_1+33+l_2+17}:\ r_2 \leftarrow r_2+1$ $\mathbf{I}_{m+4+l_1+33+l_2+18}:\ r_2 \leftarrow r_2+1$ $\mathbf{I}_{m+4+l_1+33+l_2+19}:\ r_8 \leftarrow r_{r_2}$

	$\mathbf{I}_{m+4+l_1+33+l_2+20}:\ r_3 \leftarrow r_{r_2}$
	$\mathbf{I}_{m+4+l_1+33+l_2+21}:\ r_2 \leftarrow r_2 + r_3$
$r_{10} \leftarrow r_{r_1+2+r_{r_1+2}+r_{r_1+2}+r_8}$	$\mathbf{I}_{m+4+l_1+33+l_2+22}:\ r_2 \leftarrow r_2 + r_3$
	$\mathbf{I}_{m+4+l_1+33+l_2+23}:\ r_2 \leftarrow r_2 + r_8$
	$\mathbf{I}_{m+4+l_1+33+l_2+24}:\ r_{10} \leftarrow r_{r_2}$
$r_8 \leftarrow r_8 \dot{-} 1$	$\mathbf{I}_{m+4+l_1+33+l_2+25}:\ r_8 \leftarrow r_8 \dot{-} 1$
	$\mathbf{I}_{m+4+l_1+33+l_2+26}:\ r_9 \leftarrow 0$
	$\mathbf{I}_{m+4+l_1+33+l_2+27}:$ **si** $r_8 = r_9$ **entonces** **ir a** instrucción $m+4+l_1+33+l_2+35+l_3+3$ **en otro caso** **ir a** instrucción siguiente
mientras $r_8 > 0$	
	$\mathbf{I}_{m+4+l_1+33+l_2+28}:\ r_2 \leftarrow r_1$
	$\mathbf{I}_{m+4+l_1+33+l_2+29}:\ r_2 \leftarrow r_2 + 1$
	$\mathbf{I}_{m+4+l_1+33+l_2+30}:\ r_2 \leftarrow r_2 + 1$
$r_{11} \leftarrow r_{r_1+2+r_{r_1+2}+r_{r_1+2}+r_8}$	$\mathbf{I}_{m+4+l_1+33+l_2+31}:\ r_3 \leftarrow r_{r_2}$
	$\mathbf{I}_{m+4+l_1+33+l_2+32}:\ r_2 \leftarrow r_2 + r_3$
	$\mathbf{I}_{m+4+l_1+33+l_2+33}:\ r_2 \leftarrow r_2 + r_3$
	$\mathbf{I}_{m+4+l_1+33+l_2+34}:\ r_2 \leftarrow r_2 + r_8$
	$\mathbf{I}_{m+4+l_1+33+l_2+35}:\ r_{11} \leftarrow r_{r_2}$
$((\mathbf{Q}_3^{[9]})^{(9)})^*$	$((\mathbf{Q}_3^{[9]})^{(9)})^*$
$r_8 \leftarrow r_8 \dot{-} 1$	$\mathbf{I}_{m+4+l_1+33+l_2+35+l_3+1}:\ r_8 \leftarrow r_8 \dot{-} 1$
	$\mathbf{I}_{m+4+l_1+33+l_2+35+l_3+2}:$ **si** $r_1 = r_1$ **entonces** **ir a** instrucción $m+4+l_1+33+l_2+27$ **en otro caso** **ir a** instrucción siguiente
fin mientras	
$r_1 \leftarrow r_{10}$	$\mathbf{I}_{m+4+l_1+33+l_2+35+l_3+3}:\ r_1 \leftarrow r_{10}$

A partir de los detalles que acompañan a la descripción del programa $\mathbf{P}$ podemos observar que para cualquier tupla de elementos $(x_1, ..., x_n)$, como el

programa anterior siempre se detiene, tenemos que siempre existe $t \in \omega$ tal que $Ctrl_{\mathbf{P}}^{t+1}((0,..,0,\underbrace{x_1}_{m},...,x_n),1) = y_1$ donde $y_1 = \prod_{i=1}^{n} p(i)^{x_i+1}$. Por lo tanto la función configuración es calculable.

Detalles de la proposición 1.3.2

En este apartado damos más detalles de las demostraciones de los incisos *(i)*, *(ii)*, *(iii)* de la proposición 1,3,2.

Detalles del inciso *(i)*

Comenzamos con los detalles que corresponden al inciso *(i)*. Debemos demostrar que la función $\boldsymbol{\varphi}_{\mathbf{2}}^{-1}$ es 2-calculable. Sea **P** el siguiente programa.

Programa en pseudocódigo	**Explicación del programa**
si $r_1 \neq 0$ **entonces**	Si la entrada del programa es distinta de cero se ejecuta lo siguiente
$r_2 \leftarrow 1$	Inicializamos un contador cuyo contenido será el número 1 y nos servirá para llevar la suma de los primeros n números naturales.
$r_3 \leftarrow 1$	Inicializamos un contador cuyo contenido será el número 1 y nos servirá para llevar el número n que le ha sido añadido a la suma en un momento dado.
mientras $r_2 \leq r_1$	Inicializamos un ciclo en el que realizaremos la suma de los primeros números naturales y nos detendremos cunaod esta suma sobrepase a la entrada inicial.
$r_3 \leftarrow r_3 + 1$	Aumentamos en uno el número natural que vamos a añadir a la suma.
$r_2 \leftarrow r_2 + r_3$	Realizamos la suma de dcho número con lo que llevamos sumado hasta el momento.

fin mientras	Finalizamos el ciclo si la suma ha superado a la entrada inicial.
$r_2 \leftarrow r_2 \dot{-} r_3$	A la suma que obtuvimos le restamos el número natural que hizo que ésta sobrepasara a la entrada inicial.
$r_3 \leftarrow r_3 \dot{-} 1$	Restamos uno al número natural que hizo sobrepasar la suma para obtener al menor número que hace que la suma es menor o igual que la entrada inicial.
$r_4 \leftarrow r_1 \dot{-} r_2$	A la entrada le restamos la suma para obtener el valor de la primera coordenada que el programa va a devolver.
$r_5 \leftarrow r_3 \dot{-} r_4$	Al mayor número natural que hace que la suma no sobrepase a la entrada inicial le restamos el valor obtenido en el paso anterior.
$r_1 \leftarrow r_4$	Preparaamos el primer resultado para ser devuelto por el programa.
$r_2 \leftarrow r_5$	Preparamos el segundo resultado para ser devuelto por el programa.
$r_3 \leftarrow 0$	Hacemos que los registros 3 al 8 tengan como contenido cero para que la tupla a ser devuelta tenga la forma que se requiere en la definición de calculabilidad.
$r_4 \leftarrow 0$	
$r_5 \leftarrow 0$	
$r_6 \leftarrow 0$	
$r_7 \leftarrow 0$	
$r_8 \leftarrow 0$	

en otro caso	Si la entrada inicial es 0 entonces se ejecuta lo siguiente.
$r_2 \leftarrow 0$	Basta devolver en la segunda coordenada el valor 0.
fin si	Se finaliza la condicional.

Ahora traduciremos el programa anterior escrito en pseudocódigo a instrucciones RAM para ver que en efecto es un programa.

Programa en pseudocódigo	**Programa en instrucciones RAM**
si $r_1 \neq 0$ **entonces**	$\mathbf{I}_1:\ r_6 \leftarrow 0$ $\mathbf{I}_2:$ **si** $r_1 = r_6$ **entonces** **ir a** instrucción número 46 **en otro caso** **ir a** instrucción siguiente
$r_2 \leftarrow 1$	$\mathbf{I}_3:\ r_2 \leftarrow 0$ $\mathbf{I}_4:\ r_2 \leftarrow r_2 + 1$
$r_3 \leftarrow 1$	$\mathbf{I}_5:\ r_3 \leftarrow 0$ $\mathbf{I}_6:\ r_3 \leftarrow r_3 + 1$
mientras $r_2 \leq r_1$	$\mathbf{I}_7:\ r_7 \leftarrow r_7 + 1$ $\mathbf{I}_8:$ **si** $r_7 = r_6$ **entonces** **ir a** instrucción número 10 **en otro caso** **ir a** instrucción siguiente $\mathbf{I}_9:$ **si** $r_1 = r_1$ **entonces** **ir a** instrucción número 19 **en otro caso** **ir a** instrucción siguiente
$r_3 \leftarrow r_3 + 1$	$\mathbf{I}_{10}:\ r_3 \leftarrow r_3 + 1$
$r_2 \leftarrow r_2 + r_3$	$\mathbf{I}_{11}:\ r_2 \leftarrow r_2 + r_3$

	$\mathbf{I}_{12}:\ r_7 \leftarrow r_2$
	$\mathbf{I}_{13}:\ r_8 \leftarrow 0$
	$\mathbf{I}_{14}:$ **si** $r_8 = r_1$ **entonces**
	ir a instrucción número 18
	en otro caso
	ir a instrucción siguiente
	$\mathbf{I}_{15}:\ r_7 \leftarrow r_7 \dot{-} 1$
	$\mathbf{I}_{16}:\ r_8 \leftarrow r_8 + 1$
fin mientras	$\mathbf{I}_{17}:$ **si** $r_1 = r_1$ **entonces**
	ir a instrucción número 14
	en otro caso
	ir a instrucción siguiente
	$\mathbf{I}_{18}:$ **si** $r_1 = r_1$ **entonces**
	ir a instrucción número 8
	en otro caso
	ir a instrucción siguiente
	$\mathbf{I}_{19}:\ r_8 \leftarrow 0$
	$\mathbf{I}_{20}:$ **si** $r_8 = r_3$ **entonces**
	ir a instrucción número 24
	en otro caso
	ir a instrucción siguiente
$r_2 \leftarrow r_2 \dot{-} r_3$	$\mathbf{I}_{21}:\ r_2 \leftarrow r_2 \dot{-} 1$
	$\mathbf{I}_{22}:\ r_8 \leftarrow r_8 + 1$
	$\mathbf{I}_{23}:$ **si** $r_1 = r_1$ **entonces**
	ir a instrucción número 20
	en otro caso
	ir a instrucción siguiente
$r_3 \leftarrow r_3 \dot{-} 1$	$\mathbf{I}_{24}:\ r_3 \leftarrow r_3 \dot{-} 1$
	$\mathbf{I}_{25}:\ r_4 \leftarrow r_1$
	$\mathbf{I}_{26}:\ r_8 \leftarrow 0$
	$\mathbf{I}_{27}:$ **si** $r_8 = r_2$ **entonces**
	ir a instrucción número 31
	en otro caso
	ir a instrucción siguiente
$r_4 \leftarrow r_1 \dot{-} r_2$	$\mathbf{I}_{28}:\ r_4 \leftarrow r_4 \dot{-} 1$
	$\mathbf{I}_{29}:\ r_8 \leftarrow r_8 + 1$
	$\mathbf{I}_{30}:$ **si** $r_1 = r_1$ **entonces**
	ir a instrucción número 27
	en otro caso
	ir a instrucción siguiente

	$\mathbf{I}_{31}:\ r_5 \leftarrow r_3$ $\mathbf{I}_{32}:\ r_8 \leftarrow 0$ $\mathbf{I}_{33}:$ **si** $r_8 = r_4$ **entonces** **ir a** instrucción número 37 **en otro caso** **ir a** instrucción siguiente
$r_5 \leftarrow r_3 \dot{-} r_4$	$\mathbf{I}_{34}:\ r_5 \leftarrow r_5 \dot{-} 1$ $\mathbf{I}_{35}:\ r_8 \leftarrow r_8 + 1$ $\mathbf{I}_{36}:$ **si** $r_1 = r_1$ **entonces** **ir a** instrucción número 33 **en otro caso** **ir a** instrucción siguiente
$r_1 \leftarrow r_4$	$\mathbf{I}_{37}:\ r_1 \leftarrow r_4$
$r_2 \leftarrow r_5$	$\mathbf{I}_{38}:\ r_2 \leftarrow r_5$
$r_3 \leftarrow 0$	$\mathbf{I}_{39}:\ r_3 \leftarrow 0$
$r_4 \leftarrow 0$	$\mathbf{I}_{40}:\ r_4 \leftarrow 0$
$r_5 \leftarrow 0$	$\mathbf{I}_{41}:\ r_5 \leftarrow 0$
$r_6 \leftarrow 0$	$\mathbf{I}_{42}:\ r_6 \leftarrow 0$
$r_7 \leftarrow 0$	$\mathbf{I}_{43}:\ r_7 \leftarrow 0$
$r_8 \leftarrow 0$	$\mathbf{I}_{44}:\ r_8 \leftarrow 0$
en otro caso	$\mathbf{I}_{45}:$ **si** $r_1 = r_1$ **entonces** **ir a** instrucción número 47 **en otro caso** **ir a** instrucción siguiente
$r_2 \leftarrow 0$	$\mathbf{I}_{46}:\ r_2 \leftarrow 0$
fin si	

Por las observaciones que se hicieron en la descripción del programa $\mathbf{P}$ tenemos que para cualquier $x \in \omega$, existe $t \in \omega$ tal que $Ctrl^t_{\mathbf{P}}(x,1) = ((x_1, x_2, 0, ..., \underbrace{0}_{8}), k+1)$ donde $\boldsymbol{\varphi}^{-1}_{\mathbf{2}}(x) = (x_1, x_2)$. Por lo tanto la función $\boldsymbol{\varphi}^{-1}_{\mathbf{2}}$ es 2-calculable.

Detalles del inciso *(ii)*

Ahora mostraremos los detalles de la demostración inciso *(ii)* en la misma proposición. Para concluir la demostración faltaba ver que $\boldsymbol{\varphi}_{\boldsymbol{n+1}}^{-1}$ es $n+1$-calculable. A continuación mostramos un programa que hace $\boldsymbol{\varphi}_{\boldsymbol{n+1}}^{-1}$ una función $n+1$-calculable.

Sean $\mathbf{Q}$ el programa que hace a $\boldsymbol{\varphi}_{\boldsymbol{2}}^{-1}$ 2-calculable. Si $n+2<11$ definimos a $\mathbf{P}$ como el siguiente programa.

Programa en pseudocódigo	**Explicación del programa**
$r_9 \leftarrow 1$	Inicia un contador que llevará la cantidad de aplicaciones de $\boldsymbol{\varphi}_{\boldsymbol{2}}^{-1}$.
mientras $r_9 \leq n$	Se ejecutará un ciclo en el que se relizará la aplicación del programa que hace a $\boldsymbol{\varphi}_{\boldsymbol{2}}^{-1}$ una función 2-calculable n-veces sobre la entrada inicial y sobre los resultados que vayamos obteniendo.
Q	Se ejecuta el programa mencionado anteriormente.
$r_{10+r_9} \leftarrow r_1$	El primer resultado obtenido se guarda en un registro posterior aal registro auxiliar r_{10}.
$r_9 \leftarrow r_9 + 1$	Aumentamos en uno el contador que lleva la cantidad de aplicaciones de $\boldsymbol{\varphi}_{\boldsymbol{2}}^{-1}$.
$r_1 \leftarrow r_2$	El resultado de la segunda coordenada se utilizará para una nueva aplicación de $\boldsymbol{\varphi}_{\boldsymbol{2}}^{-1}$.
fin mientras	Al finalizar el ciclo habremos obtenido $n+1$-coordenadas.
$r_{10+n+1} \leftarrow r_2$	Tras este paso habremos guardado todas las coordenadas en registros determinados.

$r_1 \leftarrow r_{11}$	Colocamos la primer coordenada en posición para ser devuelta por el programa.
$r_{11} \leftarrow 0$	Hacemos a cero el contenido de r_{11} para obtener una tupla final de la forma que requiere la definición de calculabilidad.
$r_2 \leftarrow r_{12}$	Repetimos lo mismo con la segunda coordenada obtenida.
$r_{12} \leftarrow 0$	Realizamos el mismo proceso con el registro r_{12}.
$\vdots$	
$r_{n+1} \leftarrow r_{10+n+1}$	Tras este paso ya tendremos a todas las coordenadas en posición de ser devueltas por el programa.
$r_{10+n+1} \leftarrow 0$	
$r_{n+2} \leftarrow 0$	Nos aseguramos de que los registros que se encuentran después del registro $n+1$ tengan como contenido 0.
$\vdots$	
$r_{n+9} \leftarrow 0$	

A continuación desarrollamos el programa **P** de modo que sea más sencillo notar que se trata de un programa en instrucciones RAM.

Programa en pseudocódigo	**Programa en instrucciones RAM**
$r_9 \leftarrow 1$	$\mathbf{I}_1:\ r_9 \leftarrow 0$ $\mathbf{I}_2:\ r_9 \leftarrow r_9 + 1$

	$\mathbf{I}_3:\ r_{10} \leftarrow 0$ $\mathbf{I}_4:\ r_{10} \leftarrow r_{10}+1$ $\vdots$ $\mathbf{I}_{3+n}:\ r_{10} \leftarrow r_{10}+1$ } n
mientras $r_9 \leq n$	$\mathbf{I}_{3+n+1}:$ **si** $r_9 = r_{10}$ **entonces** **ir a** instrucción número $3+n+1+l+17$ **en otro caso** **ir a** instrucción siguiente
Q	**Q**
$r_{10+r_9} \leftarrow r_1$	$\mathbf{I}_{3+n+1+l+1}:\ r_3 \leftarrow 0$ $\mathbf{I}_{3+n+1+l+2}:\ r_3 \leftarrow r_3+1$ $\vdots$ $\mathbf{I}_{3+n+1+l+11}:\ r_3 \leftarrow r_3+1$ } 10 $\mathbf{I}_{3+n+1+l+12}:\ r_3 \leftarrow r_3+r_9$ $\mathbf{I}_{3+n+1+l+13}:\ r_{r_3} \leftarrow r_1$
$r_9 \leftarrow r_9+1$	$\mathbf{I}_{3+n+1+l+14}:\ r_9 \leftarrow r_9+1$
$r_1 \leftarrow r_2$	$\mathbf{I}_{3+n+1+l+15}:\ r_1 \leftarrow r_2$
fin mientras	$\mathbf{I}_{3+n+1+l+16}:$ **si** $r_1 = r_1$ **entonces** **ir a** instrucción número $3+n+1$ **en otro caso** **ir a** instrucción siguiente
$r_{10+n+1} \leftarrow r_2$	$\mathbf{I}_{3+n+1+l+17}:\ r_{10+n+1} \leftarrow r_2$
$r_1 \leftarrow r_{11}$	$\mathbf{I}_{3+n+1+l+18}:\ r_1 \leftarrow r_{11}$
$r_1 \leftarrow 0$	$\mathbf{I}_{3+n+1+l+19}:\ r_1 \leftarrow r_0$
$r_2 \leftarrow r_{12}$	$\mathbf{I}_{3+n+1+l+20}:\ r_2 \leftarrow r_{12}$
$r_2 \leftarrow 0$	$\mathbf{I}_{3+n+1+l+21}:\ r_2 \leftarrow 0$

$\vdots$	
$r_{n+1} \leftarrow r_{10+n+1}$	$\mathbf{I}_{3+n+1+l+17+2n+1} :\ r_{n+1} \leftarrow r_{10+n+1}$
$r_{10+n+1} \leftarrow 0$	$\mathbf{I}_{3+n+1+l+17+2n+2} :\ r_{n+1} \leftarrow 0$
$r_{n+2} \leftarrow 0$	$\mathbf{I}_{3+n+1+l+17+2n+2+1} :\ r_{n+2} \leftarrow 0$
$\vdots$	
$r_{n+9} \leftarrow 0$	$\mathbf{I}_{3+n+1+l+17+2n+2+8} :\ r_{n+9} \leftarrow 0$

Por la descripción del programa $\mathbf{P}$, podemos obtener que para cualquier $x \in \omega$ existe $t \in \omega$ tal que $Ctrl_{\mathbf{P}}(x,1) = ((x_1,...,x_n),k+1)$ donde $\boldsymbol{\varphi}^{-1}_{\boldsymbol{n+1}}(x) = (x_1,...,x_n,0,...,\underbrace{0}_{s})$ para algún $s \in \omega$. Por lo tanto $\boldsymbol{\varphi}^{-1}_{\boldsymbol{n+1}}$ es $n+1$-calculable.

Detalles del inciso *(iii)*

Ahora demostraremos los detalles de la demostración del inciso *(iii)* de la misma proposición. Primero mostraremos que $\boldsymbol{\psi}$ es una función calculable. Sea $\mathbf{Q}$ un programa que hace a $\boldsymbol{\varphi_2}$ una función computable.

Sea $s = \rho_{\mathbf{Q}} + 11$. Definimos como $\mathbf{P}$ al siguiente programa:

Programa en pseudocódigo	**Explicación del programa**
$r_{s-1} \leftarrow r_s$	Guardamos en un registro la cantidad de elementos de la tupla dada inicialmente.
$r_1 \leftarrow s \dot{-} 2$	Guardamos en el primer registro la locación final del espacio de trabajo.
$r_8 \leftarrow r_{r_1+2}$	Utilizaremos como contador a r_8 que contendrá primero a la cantidad de elementos de la tupla inicial.
mientras $r_8 > 1$	Iniciaremos un ciclo en el que aplicaremos el programa que hace a $\boldsymbol{\varphi_2}$ computable a cada par de la tupla inicial comenzando de derecha a izquierda y sobre los resultados que vayamos obteniendo.

$r_{10} \leftarrow r_{r_1+1+r_8}$	Preparamos una coordenada como entrada del programa anteriormente mencionado.
$r_{11} \leftarrow r_{r_1+2+r_8}$	Preparamos una segunda coordenada para el mismo programa.
$((\mathbf{Q}^{[9]})^{(9)})^*$	Ejecutamos el programa que hace a $\boldsymbol{\varphi_2}$ una función computable.
$r_{r_1+1+r_8} \leftarrow r_{10}$	El resultado lo guardamos en un registro para que pueda ser utilizado en una ejecución más en este ciclo.
$r_8 \leftarrow r_8 \dot{-} 1$	Disminuimos en uno el contador.
fin mientras	Al finalizar el ciclo obtendremos un único número que resultará de codificar todas las coordenadas de la tupla inicial.
$r_{10} \leftarrow r_{r_1+2} \dot{-} 1$	Preparamos la cantidad de coordenadas de la tupla inicial y le rstamos uno para una ejecución del programa que hace a $\boldsymbol{\varphi_2}$ una función computable.
$r_{11} \leftarrow r_{r_1+3}$	Preparamos el resultado obtenido por el ciclo anterior para la ejecución del programa mencionado anteriormente.
$((\mathbf{Q}^{[9]})^{(9)})^*$	Ejecutamos el programa mencionado para obtener el valor de $\boldsymbol{\psi}$ en la tupla inicial dada.
$r_1 \leftarrow r_{10}$	Preparamos el resultado obtenido para que sea devuelto por el programa.

A continuación mostraremos de forma explícita que el programa anterior escrito en pseudocódigo es en efecto un programa en instrucciones RAM.

Programa en pseudocódigo	**Programa en instrucciones RAM**
$r_{s-1} \leftarrow r_s$	$\mathbf{I}_1$ $r_{s-1} \leftarrow r_s$
$r_1 \leftarrow s \dot{-} 2$	$\mathbf{I}_2$ $r_1 \leftarrow 0$ $\mathbf{I}_3$: $r_1 \leftarrow r_1 + 1$ $\vdots$ $\mathbf{I}_s$: $r_1 \leftarrow r_1 + 1$ } $s \dot{-} 2$
$r_8 \leftarrow r_{r_1+2}$	$\mathbf{I}_{s+1}$: $r_2 \leftarrow r_1$ $\mathbf{I}_{s+2}$: $r_2 \leftarrow r_2 + 1$ $\mathbf{I}_{s+3}$: $r_2 \leftarrow r_2 + 1$ $\mathbf{I}_{s+4}$: $r_8 \leftarrow r_{r_2}$
mientras $r_8 > 1$	$\mathbf{I}_{s+5}$: $r_9 \leftarrow 0$ $\mathbf{I}_{s+6}$: $r_9 \leftarrow r_9 + 1$ $\mathbf{I}_{s+7}$: **si** $r_8 = r_9$ **entonces** **ir a** instrucción número $s + 13 + l + 7$ **en otro caso** **ir a** instrucción siguiente
$r_{10} \leftarrow r_{r_1+1+r_8}$	$\mathbf{I}_{s+8}$: $r_2 \leftarrow r_1$ $\mathbf{I}_{s+9}$: $r_2 \leftarrow r_2 + 1$ $\mathbf{I}_{s+10}$: $r_2 \leftarrow r_2 + r_8$ $\mathbf{I}_{s+11}$: $r_{10} \leftarrow r_{r_2}$
$r_{11} \leftarrow r_{r_1+2+r_8}$	$\mathbf{I}_{s+12}$: $r_2 \leftarrow r_2 + 1$ $\mathbf{I}_{s+13}$: $r_{11} \leftarrow r_{r_2}$
$((\mathbf{Q}^{[9]})^{(9)})^*$	$((\mathbf{Q}^{[9]})^{(9)})^*$
$r_{r_1+1+r_8} \leftarrow r_{10}$	$\mathbf{I}_{s+13+l+1}$: $r_2 \leftarrow r_1$ $\mathbf{I}_{s+13+l+2}$: $r_2 \leftarrow r_2 + 1$ $\mathbf{I}_{s+13+l+3}$: $r_2 \leftarrow r_2 + r_8$ $\mathbf{I}_{s+13+l+4}$: $r_{r_2} \leftarrow r_{10}$
$r_8 \leftarrow r_8 \dot{-} 1$	$\mathbf{I}_{s+13+l+5}$: $r_8 \leftarrow r_8 \dot{-} 1$

fin mientras	$\mathbf{I}_{s+13+l+6}:$ **si** $r_1 = r_1$ **entonces** **ir a** instrucción número $s+7$ **en otro caso** **ir a** instrucción siguiente
$r_{10} \leftarrow r_{r_1+2} \dot{-} 1$	$\mathbf{I}_{s+13+l+7}: r_2 \leftarrow r_1$ $\mathbf{I}_{s+13+l+8}: r_2 \leftarrow r_2 + 1$ $\mathbf{I}_{s+13+l+9}: r_2 \leftarrow r_2 + 1$ $\mathbf{I}_{s+13+l+10}: r_{10} \leftarrow r_{r_2}$ $\mathbf{I}_{s+13+l+11}: r_{10} \leftarrow r_{10} \dot{-} 1$
$r_{11} \leftarrow r_{r_1+3}$	$\mathbf{I}_{s+13+l+12}: r_2 \leftarrow r_2 + 1$ $\mathbf{I}_{s+13+l+13}: r_{11} \leftarrow r_{r_2}$
$((\mathbf{Q}^{[9]})^{(9)})^*$	$((\mathbf{Q}^{[9]})^{(9)})^*$
$r_1 \leftarrow r_{10}$	$\mathbf{I}_{s+13+l+13+l+1}: r_1 \leftarrow r_{10}$

Observemos que para cualesquiera elementos $x_1, ..., x_n \in \omega$ se cumple que existe $t \in \omega$ tal que $Ctrl_{\mathbf{P}}^{t+1}((0, ..., 0, \underbrace{n}_{s}, x_1, ..., x_n), 1) = y$ donde $\boldsymbol{\psi}(x_1, ..., x_n) = y$. Por lo tanto $\boldsymbol{\psi}$ es calculable.

Ahora veamos que la función $\boldsymbol{\psi}^{-1}$ es calculable. Para ello sea $\mathbf{Q}$ el programa que hace a $\boldsymbol{\varphi}_{\mathbf{2}}^{-1}$ una función 2-calculable. Definimos como $\mathbf{P}$ al siguiente programa:

Programa en pseudocódigo	**Explicación del programa**
Q	Ejecutamos el programa $\mathbf{Q}$ en la entrada inicial y se obtendrán dos números; el primero sumado más uno es la longitud de la tupla que debemos obtener, el segundo es la codificación de la tupla que debemos obtener.
$r_9 \leftarrow r_1$	Guardamos la cantidad de elementos a obtener menos uno para utilizarlo como contador.
$r_9 \leftarrow r_9 + 1$	Sumamos uno para obtener la cantidad total de elementos a obtener.

$r_{12} \leftarrow r_9$	Guardamos la cantidad anteriormente mencionada.
$r_{13} \leftarrow r_2$	Guardamos la segunda coordenada obtenida.
$r_1 \leftarrow r_2$	Preparamos dicho resultado para una posible nueva ejecución del programa que hace a $\boldsymbol{\varphi_2^{-1}}$ una función 2-calculable.
$r_{11} \leftarrow 1$	Inicizalizamos un contador que nos ayudará a colocar en orden las coordenadas que vayamos obteniendo.
mientras $r_9 > 1$	Iniciamos un ciclo en el que iremos hallando las coordenadas de la tupla que debemos obtener.
Q	Se ejecuta nuevamente el programa que hace a $\boldsymbol{\varphi_2^{-1}}$ una función 2-calculable.
$r_{12+r_{11}} \leftarrow r_1$	Guardamos el primer resultado que obtengamos para o bien ser utilizado en una nueva iteración del programa mencionado anteriormente o bien para ser considerado como un elemento de la tupla que debemos obtener.
$r_{13+r_{11}} \leftarrow r_2$	Guardamos el segundo resultado por la misma razón del resultado anterior.
$r_1 \leftarrow r_2$	Preparamos el segundo resultado para una nueva ejecución del programa mencionado anteriormente.
$r_9 \leftarrow r_9 \dot{-} 1$	Disminuimos en uno el contador que lleva las veces de la aplicación del programa **Q**.
$r_{11} \leftarrow r_{11} + 1$	Aumentamos en uno el contador que lleva la posición para que los elementos que vamos obteniendo se guarden.
fin mientras	Al finalizar este ciclo habremos obtenido la cantidad de elementos que debíamos obtener y las coordenadas de la tupla que requeriamos.

$r_1 \leftarrow 0$	Hacemos a 0 los primeros registros para llegar a la forma que pide la definición de función calculable.
$\vdots$	
$r_{11} \leftarrow 0$	Tras este paso habremos obtenido el resultado que requiere la definición.

Ahora veamos que el programa anterior está en efecto escrito con instrucciones RAM.

Programa en pseudocódigo	**Programa en instrucciones RAM**
Q	**Q**
$r_9 \leftarrow r_1$	$\mathbf{I}_{l+1}:\ r_9 \leftarrow r_1$
$r_9 \leftarrow r_9 + 1$	$\mathbf{I}_{l+2}:\ r_9 \leftarrow r_9 + 1$
$r_{12} \leftarrow r_9$	$\mathbf{I}_{l+3}:\ r_{12} \leftarrow r_9$
$r_{13} \leftarrow r_2$	$\mathbf{I}_{l+4}:\ r_{13} \leftarrow r_2$
$r_1 \leftarrow r_2$	$\mathbf{I}_{l+5}:\ r_1 \leftarrow r_2$
$r_{11} \leftarrow 1$	$\mathbf{I}_{l+6}:\ r_{11} \leftarrow 0$ $\mathbf{I}_{l+7}:\ r_{11} \leftarrow r_{11} + 1$
mientras $r_9 > 1$	$\mathbf{I}_{l+8}:\ r_{10} \leftarrow 0$ $\mathbf{I}_{l+9}:\ r_{10} \leftarrow r_{10} + 1$ $\mathbf{I}_{l+10}:$ **si** $r_9 = r_{10}$ **entonces** **ir a** instrucción número $l + 10 + l + 22$ **en otro caso** **ir a** instrucción siguiente
Q	**Q**

	$\mathbf{I}_{l+10+l+1}:\ r_3 \leftarrow 0$ $\mathbf{I}_{l+10+l+2}:\ r_3 \leftarrow r_3+1$ $\vdots$ $\mathbf{I}_{l+10+l+13}:\ r_3 \leftarrow r_3+1$ $\mathbf{I}_{l+10+l+14}:\ r_3 \leftarrow r_3+r_{11}$ $\mathbf{I}_{l+10+l+15}:\ r_{r_3} \leftarrow r_1$	12
$r_{12+r_{11}} \leftarrow r_1$		
$r_{13+r_{11}} \leftarrow r_2$	$\mathbf{I}_{l+10+l+16}:\ r_3 \leftarrow r_3+1$ $\mathbf{I}_{l+10+l+17}:\ r_{r_3} \leftarrow r_2$	
$r_1 \leftarrow r_2$	$\mathbf{I}_{l+10+l+18}:\ r_1 \leftarrow r_2$	
$r_9 \leftarrow r_9 \dot{-} 1$	$\mathbf{I}_{l+10+l+19}:\ r_9 \leftarrow r_9 \dot{-} 1$	
$r_{11} \leftarrow r_{11}+1$	$\mathbf{I}_{l+10+l+20}:\ r_{11} \leftarrow r_{11}+1$	
	$\mathbf{I}_{l+10+l+21}:$ **si** $r_1 = r_1$ **entonces** **ir a** instrucción número $l+10$ **en otro caso** **ir a** instrucción siguiente	
fin mientras		
$r_1 \leftarrow 0$	$\mathbf{I}_{l+10+l+22}:\ r_1 \leftarrow 0$	
$\vdots$	$\vdots$	
$r_{11} \leftarrow 0$	$\mathbf{I}_{l+10+l+32}:\ r_{11} \leftarrow 0$	

Observemos que para todo $x \in \omega$ existe $t \in \omega$ tal que $Ctrl^t_{\mathbf{P}}(x,1) = ((0,...,0,\underbrace{n}_{12},x_1,...,x_n),k+1)$ donde $\boldsymbol{\psi}^{-1}(x) = (x_1,...,x_n)$. Por lo tanto la función $\boldsymbol{\psi}$ es calculable.

Índice alfabético

Bibliografía

[CK] Chang C.C., Keisler H.J. *Model Theory.* Elsevier Science. 3 ed. 1992. Cap 1 y 2.

[Cut] Cutland N. *An Introduction to Recursive Function Theory.* Cambridge University Press. 1 ed. 1980. Cap 1-5 y 9.

[Dav] Davis M. *Computability and Unsolvability.* Mc-Graw Hill Company Book inc. 1 ed. 1958. Cap. I-V.

[Mon] Montalbán A. *Computable Structure Theory.* Última compilación 25 Jun. 2019. Borrador. Cap. I, II, III, IV, VIII. `https://math.berkeley.edu/~antonio/CSTpart1.pdf`

[Soa] Soare R.I. *Recursively Enumerable Sets and Degrees.* Springer-Verlag. 1 ed. 1987. Cap. I, II, VI, VII y VIII.