TRABAJO FIN DE MÁSTER

Facultad de Filosofía y Letras

Máster en Lógica y Filosofía de la Ciencia

Autor: **Héctor Sanz Herranz**

Tutor: Enrique Alonso González

[*Panorámica de la teoría computacional: Desarrollo y problemas abiertos.*]

Universidad de Valladolid

Julio de 2017.

PROLEGÓMENOS.

RESUMEN.

En este trabajo se exponen sendas introducciones a los conceptos propios de la teoría de la computación y de la teoría de la complejidad computacional, prestando especial atención a los movimientos filosóficos que propiciaron su desarrollo. Asimismo, se presenta una de las incógnitas sin resolver más importantes de la Matemática, la *conjetura de Cook*, así como sus consiguientes implicaciones al respecto de la naturaleza de los propios problemas.

ABSTRACT.

In this dissertation, two introductions are presented to the concepts of computational theory and the computational complexity theory paying special attention to the philosophical movements that led up to its development. Also presents one of the most important unresolved unknows of Mathematics, the Cook's conjecture, as well as its consequent implications for the nature of the problems themselves.

TÉRMINOS CLAVE.

Clase de complejidad, ¿P = NP?, lenguaje formal, función computable, problema de decisión, programa de Hilbert, máquina de Turing.

KEY WORDS.

Complexity class, P = NP?, formal language, computable function, decision problem, Hilbert's program, Turing machine.

AGRADECIMIENTOS.

Siempre he tratado de ser prudente a la hora de salvaguardar la intimidad de quienes se encuentran a mi alrededor e, incluso, la mía propia. Por otra parte, me he dado cuenta recientemente de que, aun cuando habitualmente dedico gran cantidad de tiempo a recopilar y estudiar las distintas cuestiones que acaparan mi interés, no termino de integrarlas a mi entendimiento hasta que no las comparto con alguien, lo cual puede resultar a menudo trágico para el resto dada la extraña naturaleza de las mismas. Por ello, mi agradecimiento más sincero será para quienes han decidido en alguna ocasión soportar en carnes propias mis pensamientos, ya sea prestándose para leer cuanto escribo, escuchar cuanto hablo o preguntar lo que creen que conozco.

Contenidos.

INTRODUCCIÓN.

La comunidad matemática no ha logrado alcanzar aún un consenso respecto a cuál ha de ser la labor primordial de los matemáticos. La tendencia predominante es la de contribuir a la ingente colección de resultados que se van acumulando en cada una de las ramas de la Matemática mientras que otros, por su parte, vuelcan sus esfuerzos en tratar de reconciliar algunas de ellas. Repárese en que se ha dado en llamar a Henri Poincaré como el *último matemático universal*, evidenciando la progresiva escisión del conocimiento matemático.

En cualquier caso, lo que es cierto es que los matemáticos trabajamos con *problemas*, esto es, los analizamos, los comprendemos, los revisamos, volvemos sobre ellos e, incluso, llegamos a desarrollar sentimientos encontrados hacia ellos. Sin embargo, nos estamos olvidando de lo más importante y es que, a menudo, también los *planteamos*. En efecto, no es nada excepcional que en el transcurso de una investigación, un matemático termine por proponer más interrogantes que aquellos a los que pretendía dar respuesta en un principio. De esta manera, es natural encontrarse con situaciones como la *paradoja de Russel*, el *teorema de Fermat* o la *conjetura de Cook*, donde los problemas toman el nombre del que para algunos es su *descubridor* y para otros, su *creador*.

De esta manera, existen dos posiciones antagónicas respecto al carácter ontológico de los problemas. Una de ellas defiende que los problemas gozan de una naturaleza objetiva aunque puede que no siempre exista una vía para *acceder* a ellos. En esta línea, se acepta que tanto su existencia como sus propiedades sean independientes de los sujetos racionales, quienes quedan relegados a un segundo plano pues resultan ser completamente contingentes. De esta manera, se entiende que se *descubre* un problema cuando se logra *acceder* a él. La otra posibilidad orbita en torno a los individuos y considera que su existencia es absolutamente necesaria para la de los *problemas*. En este sentido, los problemas se *crean* gracias al proceso cognoscitivo llevado a cabo por un sujeto racional con capacidad suficiente para ello. Esta parece ser la postura que habitualmente aceptamos en nuestra vida diaria. La Real Academia de la Lengua Española define *problema* como:

1. Cuestión que se trata de aclarar.
2. Proposición o dificultad de solución dudosa.
3. Conjunto de hechos o circunstancias que dificultan la consecución de algún fin.
4. Disgusto, preocupación.
5. Planteamiento de una situación cuya respuesta desconocida debe obtenerse a través de métodos científicos.

En efecto, se presupone la existencia de un sujeto que trata de aclarar o resolver una cuestión o dificultad (1, 2), o que pretende conseguir un determinado fin (3), o que está afecto de un disgusto o preocupación (4), o que plantea una cierta situación (5). Asimismo, podemos observar cómo se han introducido de soslayo ciertas nociones como las de *solución* o *dificultad* de un problema que resultan ser propiedades de los mismos. Aunque tales nociones nos resultan familiares es necesario realizar determinadas reflexiones al respecto antes de embarcarse en el estudio de las mismas del mismo modo que conviene elegir destino antes de pretender llegar a algún sitio pues, de otro modo, se corre el riesgo de no llegar a ninguna parte.

Como estábamos diciendo, a menudo asumimos que los problemas requieren de la existencia de individuos que los planteen o que los padezcan. Esta posición acaece en nuestra vida diaria cuando concedemos, por ejemplo, que para comprender totalmente el impacto de una determinada enfermedad es preciso haberla experimentado previamente. De hecho, conviene prestar atención a la connotación negativa de la palabra "problema". De esta manera, tendríamos que reconocer que las propiedades de los problemas dependen del sujeto racional al que conciernen y, puesto que la *dificultad* o *complejidad* de un problema resulta ser una de tales propiedades, también habrá de depender del mismo. Estudiar la complejidad de acuerdo con esta postura tiene complicaciones evidentes y quizás por ello los matemáticos habitualmente rehúsan cuestionar la naturaleza objetiva de los problemas. Como si de una escultura que trasciende a su autor se tratase, los problemas perduran mucho más que sus creadores. Por eso, aparentemente uno puede desprenderse de estos y analizar únicamente sus creaciones pero eso no debe implicar el dejar de ser conscientes de su carácter ontológico.

De ahora en adelante, nos ceñiremos únicamente a *problemas matemáticos* entendiendo estos bajo la acepción 5. de la definición general de problema, pero con la salvedad de no reconocer la necesidad de la existencia de un agente racional y aceptaremos, al menos inicialmente, que los problemas matemáticos disfrutan de una naturaleza objetiva y así también sus propiedades. De esta manera, corresponde a los matemáticos la labor de proponer una taxonomía para ellos y, para que esta sea suficientemente general, se ha de atender a las propiedades comunes a todos ellos: la *computabilidad* y la *complejidad*. Sin embargo, comprender fidedignamente tales conceptos resulta ser una ardua tarea que hemos de padecer (no olvidemos que, al fin y al cabo, estamos lidiando con problemas) y es precisamente el propósito de este trabajo el tratar de contribuir a ello.

En cuanto a la estructura de esta memoria conviene señalar que consta de dos partes diferenciadas. La primera de ellas versa sobre la *teoría de la computación*, siguiendo el trazado histórico de los acontecimientos que derivaron en su aparición, para acabar exponiendo el modelo computacional por antonomasia: la *máquina de Turing*. Respecto a la segunda, cabe apuntar que pretende constituir un acercamiento a la problemática de la *teoría de la complejidad* y, por ello, se ha tratado de evitar en medida de lo posible el incurrir en demasiados tecnicismos. Aun así, no hemos podido dejar de incluir sendas secciones que definen con el rigor debido las clases de complejidad más importantes. Por último, se acaba presentando la *conjetura de Cook*, uno de los problemas matemáticos de mayor relevancia, junto con diversas reflexiones al respecto.

1

UNA INTRODUCCIÓN A LA *teoría de la computación.*

"La gente tiene estrellas que no son las mismas.
Para unos son guías, para otros luces pequeñas.
Pero para los que son sabios, son problemas."
— (*El Principito*) Antoine de Saint-Exupery.

Resumen: Se presentan los conceptos básicos de la *teoría de la computación* relatando la manera en que se ha llegado a ellos a partir de los problemas de la fundamentación de la matemática. Asimismo, se introducen distintos modelos computacionales y, en particular, se examina detalladamente el modelo de Turing.

1.1. PRELUDIO MATEMATICO.

Antes de comenzar nuestra empresa será necesario disponer de las herramientas con las que vamos a trabajar. Por ello, hemos incluido esta sección que pretende introducir los conceptos básicos que utilizaremos a lo largo del texto.

Especial importancia van a tener los conjuntos no vacíos que son, además, finitos. A tales conjuntos los denominaremos *alfabetos* y a sus elementos, *letras*. Ahora estamos en condiciones de presentar lo que en este contexto vamos a entender por *palabra*.

Definición 1. *(Palabra, lenguaje)*
Dado un alfabeto, Σ, se dice que w es una palabra *sobre Σ si, y solo si, w es una lista o cadena finita de letras de Σ. Formalmente, se representa como $w = w_1 \dots w_n$, donde $w_i \in \Sigma$, $i = 1, \dots, n$. Al número natural n se lo denomina* longitud *de la palabra, y se denota por $|w|$, mientras que al conjunto de todas las palabras de longitud exactamente n se lo denota por Σ^n. Por su parte, existe una única palabra de longitud nula, la palabra vacía, que se suele denotar por λ.*

Al conjunto de todas las palabras sobre un alfabeto Σ se lo denota por Σ^*, y es posible identificarlos con la unión disjunta

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

A los conjuntos $\mathbb{L}$ tales que $\mathbb{L} \subseteq \Sigma^*$ se los denomina lenguajes.

La noción que vamos a presentar a continuación va a resultar de vital importancia pues nos va a permitir comprender mejor el alcance de los resultados que se tratarán más adelante.

Definición 2. *(Codificar, código, codificación)*
Dados sendos alfabetos, Δ y Σ, codificar Δ *en Σ^* consiste en dar una aplicación inyectiva $c : \Delta \longrightarrow \Sigma$. En tal caso, se dice que Σ es el* código *y $c(\Delta)$ la* codificación.

Cuando no se concrete la aplicación c utilizada, denotaremos por $\llcorner x \lrcorner$ a la codificación de un cierto objeto x, es decir, identificaremos $\llcorner x \lrcorner \equiv c(x)$. Lo más habitual será codificar en código binario, esto es, tomando $\Sigma = \{0, 1\}$. Demos paso a una nueva definición.

Definición 3. *(Función booleana, asignación booleana)*
Se dice que una función ψ *es una* función booleana *de n argumentos si* $\psi : \{0,1\}^n \longrightarrow \{0,1\}$.
A cada elemento $z \in \{0,1\}^n$ *se lo llama* asignación booleana.

Sin lugar a duda, la noción que va a jugar un papel central en este ensayo va a ser la de *algoritmo*. Si bien existen actualmente numerosos debates abiertos al respecto del concepto de *algoritmo*, comúnmente se acepta que este ha de ser un procedimiento definido mediante un conjunto finito de instrucciones que aplicado a una cantidad fija de elementos *input* produce eventualmente un elemento *output*. Durante el proceso de ejecución de un algoritmo se van generando distintos elementos conforme van aplicándose cada una de las instrucciones. Este proceso puede terminar debido a que no resten instrucciones por aplicar o, en cambio, puede no finalizar nunca. Por su parte, aunque actualmente se permite que existan algoritmos que solo aplican a lo sumo una instrucción en cada etapa (*deterministas*) y otros que gozan de más de una instrucción aplicable (*no deterministas*), conviene señalar que en lo que sigue, cuando utilicemos el término *algoritmo* nos referiremos únicamente a los deterministas salvo que se indique lo contrario.

Aunque existe cierto debate al respecto, habitualmente se acepta que los *inputs* y *outputs* de un algoritmo han de ser objetos finitos. De esta manera, no todos los números reales podrían servir de *input* a un algoritmo pues existen números cuyas infinitas cifras decimales no se siguen ningún comportamiento periódico (por ejemplo, el número π).

Asimismo, podría afirmarse que cada problema matemático no es más que una relación entre dos conjuntos. El primero, $\mathfrak{A}$, habría de ser el de todos los posibles *input* mientras que el segundo, $\mathfrak{B}$, consistiría en todas las posibles soluciones o *output*. Esta idea nos dirige inevitablemente a la noción de función. En efecto, podemos entender un problema como una función parcial [1] $f : \mathfrak{A} \longrightarrow \mathfrak{B}$. Con esto en mente, se puede decir que un algoritmo materializa el problema y que guía al *procesador*, que puede ser un humano o una máquina, desde el *input* hasta el *output* mediante una secuencia finita de instrucciones. Asimismo, se dice que el procesador *computa* el problema cuando recorre tal secuencia de instrucciones.

Conviene señalar que la noción de *algoritmo*, al contrario de lo que ocurre con otros objetos matemáticos, no está libre de ambigüedades. Quizás sea consecuencia de que se puede acceder a ella siguiendo distintos planteamientos. En la sección venidera recorreremos uno que pretende plasmar la razón de ser de un algoritmo.

[1] Se dice que una función $f : A \longrightarrow B$ es una *función parcial* si se permite que f pueda no estar definida para algunos elementos de A. Si f está definida para un cierto $a \in A$, se escribe $f(a)\downarrow$; en otro caso, $f(a)\uparrow$.

1.2. LA CRISIS DE LAS MATEMÁTICAS.

Idealmente, la manera en que la matemática adquiere conocimiento nuevo se basa en el denominado *método axiomático*. Partiendo de un conjunto de enunciados básicos que se denominan *axiomas*, el conocimiento se extiende vía generación sistemática de *proposiciones* que han de ser deducidas correctamente, esto es, siguiendo unas determinadas *reglas de deducción*. Así, para alcanzar una nueva proposición válida ha de recorrerse una secuencia finita de estados a la que usualmente se denomina *prueba*. Una proposición que goza de una prueba es lo que se denomina *teorema*. Al conjunto compuesto por los axiomas y los teoremas se lo conoce como *teoría*. Así, por definición, supondremos que todas las teorías son axiomatizables.

Desde los trabajos de Euclides hasta mediados del siglo XIX se requería que los axiomas fuesen enunciados cuya validez estuviese libre de cualquier duda, esto es, que resultasen *evidentes* a partir de la experiencia. Debido a este requerimiento, tácitamente se estaba asumiendo que los axiomas debían versar sobre *ideas* [2] cuya naturaleza objetiva estaba clara. De esta manera, los axiomas no necesitaban prueba alguna pues estaban claramente ratificados por la propia realidad. Sin embargo, a lo largo del siglo XIX surgieron serias dudas al respecto de qué era evidente y qué no, lo que conllevó el abandono de los sistemas axiomáticos basados en la evidencia en beneficio de aquellos cuyos axiomas eran aceptados como meras *hipótesis*. Estos nuevos sistemas axiomáticos no cuestionaban en absoluto la naturaleza de los objetos que trataban. Así, por ejemplo, la axiomática de Peano propuesta en 1889 describía relaciones y propiedades relativas a los números naturales a partir de nociones más básicas sin entrar a valorar qué realidad material representa un numero natural.

Una teoría erigida sobre un sistema axiomático hipotético resultó ser la *teoría de conjuntos* propuesta por el célebre matemático alemán Georg Cantor a finales del siglo XIX. El desarrollo de la teoría de conjuntos provocó que en la comunidad matemática cundiese el escepticismo. La reinterpretación del concepto de *infinito* trajo consigo numerosos resultados sumamente contraintuitivos y de lo más inesperados. Además, se había de permitir la extrapolación de razonamientos finitos al terreno de lo infinito. De hecho, el propio Cantor distinguía entre las nociones de *infinito actual* e *infinito potencial*. Por si fuera poco, en torno al año 1900, empezó a descubrirse que la teoría de conjuntos estaba afecta de numerosas *paradojas* [3].

A partir de ese momento quedó claro que había que hacer algo para combatir las paradojas que habían ido apareciendo. El objetivo era realizar las modificaciones necesarias para lograr que se asentasen los fundamentos de las matemáticas y de otros métodos axiomáticos pero esta vez con garantías de no adolecer de paradojas. Las propuestas y críticas que se realizaron fueron substancialmente diferentes y, de todas ellas,

[2] De acuerdo con el punto de vista *platónico*, existe objetivamente un mundo no material en el que habitan las *ideas* abstractas y que es *accesible* únicamente a través de nuestro intelecto.

[3] Una paradoja es la situación en la que se alcanza una conclusión inaceptable a la que se llega siguiendo razonamientos aparentemente válidos. Las paradojas más conocidas que se descubrieron fueron las de Cantor, Russel y Burali-Forti.

las más importantes dieron lugar a tres corrientes de pensamiento: el *intuicionismo*, el *logicismo* y el *formalismo*.

El intuicionismo abogaba por una matemática fundamentada sobre la noción de *prueba*, apostando por una perspectiva no platonista en la que la existencia de los objetos estaba inherentemente ligada a la existencia de su constructibilidad mental. A partir de este principio, el intuicionismo logró recomponer algunas partes de la matemática clásica demostrando, además, que estaba desprovista de paradojas. Desafortunadamente, gran parte de las matemáticas no pudieron ser adaptadas a los principios intuicionistas.

Por su parte, el propósito del logicismo iba mucho más lejos pues este ansiaba fundamentar todas las matemáticas en la lógica. Los máximos exponentes de esta corriente fueron George Boole, Gottlob Frege, Guiseppe Peano, Bertrand Russell y Alfred Whitehead. Boole estaba preocupado por analizar rigurosamente los procesos de deducción lógica que venían empleándose desde Aristóteles. Sus investigaciones dimanaron en lo que se conoce actualmente como *cálculo proposicional*. Frege, por su parte, trató de demostrar que la aritmética era deducible a partir de la lógica. Concretamente, Frege planeaba definir las nociones propias de la aritmética valiéndose de las de la lógica y, al mismo tiempo, deducir los axiomas de la aritmética a partir de los de la lógica. Simultáneamente, Peano desarrolló otro lenguaje simbólico con capacidad suficiente para expresar las proposiciones propias de la matemática que incorporaba símbolos nuevos tales como $\in$. Se puede decir que entre Frege y Peano sentaron las bases de lo que hoy se conoce como *lógica de primer orden*. La meta a la que aspiraban Russell y Whitehead era aún más pretenciosa pues querían deducir absolutamente todas las matemáticas a partir de la lógica. Para evitar las paradojas o, al menos, algunas clases de paradojas, desarrollaron lo que se conoce como *teoría de tipos*, cuya exposición se recoge en su insigne obra, los *Principia Mathematica*.

Por otro lado, los formalistas no podían aceptar la radicalidad de las medidas adoptadas por los intuicionistas. Creían que era preciso conservar toda la matemática clásica pues, pese a todo, había demostrado su valía en innumerables ocasiones. Para lograrlo, los formalistas centraron su atención en el aspecto diametralmente opuesto al significado, la *semántica*, de las expresiones matemáticas, esto es, en la *sintaxis*. El formalismo fue iniciado por el que era, junto con Henri Poincaré, el matemático más famoso del momento, David Hilbert. En su opinión, las nociones sintácticas constituían la única parte de las matemáticas que estaba libre de situaciones problemáticas, concretamente, la lógica elemental y cierta parte de la aritmética.

La idea principal del formalismo radicaba en el hecho de que, si bien numerosos conceptos matemáticos adolecían de ambigüedades que acababan por conducir a paradojas, es cierto que las nociones matemáticas siempre habían sido expresadas mediante *palabras* de algún lenguaje, bien sirviéndose de un lenguaje natural o de uno simbólico. El formalismo advirtió que las palabras no son más que una colección de *símbolos* (repárese en que esta idea originó la definición actual de *palabra* que se acepta hoy en día en el ámbito de las matemáticas (Definición 1)) y que estos estaban libres de ambigüedad pues su comprensión es inmediata una vez que han sido reconocidos. De esta forma, el entendimiento de los símbolos resulta independiente de la semántica

de las palabras. Por lo tanto, los formalistas pretendieron concebir las palabras como meras sucesiones de símbolos libres de significado. Más aún, puesto que las oraciones son sucesiones de palabras, también estas habrían de ser tratadas como secuencias de símbolos. La razón de esta postura se debe a que la sintaxis siempre goza del rigor necesario para no estar afecta de imprecisiones. Finalmente, incluso una prueba deductiva podría ser tratada como una secuencia finita de constructos del lenguaje completamente libre de vaguedades y de las consiguientes paradojas.

1.3. EL PROGRAMA DE HILBERT.

El *programa de Hilbert* constituyó un intento de reconstruir las matemáticas utilizando sistemas axiomáticos formales que no originen paradojas. Para poder comprenderlo con exactitud es preciso asimilar antes los problemas que trataban de solventarse. Tales problemas son los conocidos como *problemas de la fundamentación de la matemática* y su abordaje reveló la importancia del concepto de *algoritmo* provocando asimismo el nacimiento de la *teoría de la computación*.

El primero de los problemas que pretendían tratarse era el de la *consistencia*. Supongamos que **T** es una teoría y que $\alpha \in \mathbf{T}$ tal que tanto α como $\neg\alpha$ son deducibles en **T**. Inmediatamente se sigue que la fórmula contradictoria $\alpha \wedge \neg\alpha$ es deducible. En tal caso, se dice que la teoría **T** es *inconsistente*. En el caso contrario, se dice que la teoría **T** es *consistente*. Puesto que una teoría inconsistente carece de interés debido a que permite deducir cualquier fórmula, lo sensato es pretender alcanzar una teoría que sea consistente. Por ello, también se dice que una teoría consistente es aquella para la que existe una fórmula que no es un teorema.

Otro de los problemas era el de la *completitud sintáctica*. Supongamos que **T** es una teoría consistente y sea $\alpha \in \mathbf{T}$. Puesto que es consistente, α y $\neg\alpha$ no pueden ser simultáneamente deducibles en **T**. Pero nada impide en principio que ninguna de las dos sea deducible. En ese caso, se dice que α es *independiente* de **T**. Así, cuando al menos una de entre α y $\neg\alpha$ es deducible, se dice que la teoría **T** es *sintácticamente completa*.

El siguiente problema que se plateaba, y que es el que más atañe a las pretensiones de este ensayo, era el de la *decidibilidad*. Puesto que estamos entendiendo las teorías como las clausuras deductivas de conjuntos de axiomas, si una teoría es consistente y sintácticamente completa, a la fuerza habrá de ser también decidible. Volveremos sobre ello más adelante. Por su parte, la deducción de α o $\neg\alpha$ puede ser realmente compleja. Lo deseable sería que existiese un procedimiento finito, un algoritmo, que nos dijese si una fórmula α es deducible en **T**. A esta situación en la que uno se cuestiona la deducibilidad de α es lo que se conoce como *problema de decisión*. Cuando existe un método que permite responder efectivamente a todos los problemas de decisión que se pueden plantear en una teoría **T** se dice que **T** es *decidible*.

El último de los problemas fue el de la *completitud semántica*. Sea **T** una teoría

consistente. Se dice que una teoría es *correcta* si se verifica que

$$\vdash_{\mathbf{T}} \alpha \implies \models_{\mathbf{T}} \alpha \quad [4] \tag{1.1}$$

para cualquier α. Notemos que esto es lo mínimo que le pedimos a una teoría, pues nunca aceptaríamos que pudiésemos probar algo que no sea cierto. El recíproco de (1.1) afirma que si una proposición es verdadera, entonces es deducible en la teoría, lo cual se formaliza como

$$\models_{\mathbf{T}} \alpha \implies \vdash_{\mathbf{T}} \alpha. \tag{1.2}$$

Cuando una teoría verifica (1.2) se dice que es *semánticamente completa.* La completitud semántica quizás sea la propiedad más importante de cuantas podemos exigirle a una teoría pues afirma que aquello que verdadero en una teoría también ha de ser un teorema de la misma.

Ahora que hemos expuesto los problemas de la fundamentación de la matemática estamos en condiciones de presentar en qué consistía el *programa de Hilbert.* Señalemos que la presentación que vamos a realizar está traducida a términos actuales.

Definición 4. *(Programa de Hilbert)*
El programa de Hilbert *consistía en una empresa matemática que pretendía:*

1. *encontrar un sistema axiomático formal* ***H*** *capaz de deducir todos los teoremas de la matemática;*
2. *demostrar que* ***H*** *es consistente;*
3. *demostrar que* ***H*** *es semánticamente completo;*
4. *encontrar un algoritmo que resuelva los problemas de decisión de* ***H****.*

El último de los propósitos descritos en la definición anterior es lo que Hilbert denominó *Entscheidungsproblem.* Él creía que si se lograba completar su programa, todas las matemáticas podrían desarrollarse mecánicamente pues cualquier proposición podría ser expresada mediante una fórmula α de **H** (existencia de **H**), α sería una verdad de la teoría si, y sólo si, fuese un teorema (completitud semántica). Como, además, solo una de α y $\neg\alpha$ sería un teorema (consistencia), bastaría acudir al algoritmo que el

[4] El formalismo introduce esta notación la cual refleja lo siguiente: $\vdash_{\mathbf{T}} \alpha$ indica que α es un teorema de la teoría **T**; $\models_{\mathbf{T}} \alpha$ expresa que la fórmula α es verdadera en todo modelo de la teoría **T**.

Entscheidungsproblem dice que debería existir para dirimir cuál de ambas gozaría de tal categoría.

Inmediatamente después de que Hilbert propusiera su programa, los investigadores se pusieron manos a la obra con objeto de satisfacerlo. Si bien es cierto que los resultados relativos a 1. y 4. resultaban alentadores, pocos años después Gödel, al amparo de sus *teoremas de incompletitud*, acabaría por truncar el sueño de Hilbert tan solo unos días después de que este pronunciase su célebre *"debemos saber y sabremos"* [4]. Pero, parafraseando a Michael Ende, esa es otra historia que debe ser contada en otra ocasión.

Volviendo sobre el *Entscheidungsproblem* cabe señalar que su propósito no es otro que el de diseñar un método capaz de decidir si una fórmula arbitraria α es deducible en **H**. Como apuntamos en la sección anterior, los formalistas concebían las pruebas como meras secuencias finitas de símbolos, desarrolladas de acuerdo a unas determinadas reglas sintácticas. Por ello, un primer procedimiento podría ser el siguiente.

Procedimiento 1 : Búsqueda de una deducción de α.

Entrada: Una fórmula α.
Salida: **cierto** si se encuentra una prueba para α en **H**.

1: Generar una secuencia finita de símbolos, x.
2: **mientras** x no es una prueba de α **hacer**
3: Generar una nueva secuencia de símbolos $y \longrightarrow x$.
4: **si** x es una prueba de α. **entonces**
5: **devolver cierto**
6: **fin si**
7: **fin mientras**
8: **devolver cierto**

Debemos reparar que si la fórmula α es deducible en **H**, entonces el Procedimiento 1 parece ser una forma razonable de encontrar efectivamente una prueba para la misma. Sin embargo, si α no resulta ser deducible, entonces el procedimiento previo no terminaría nunca pues continuaría indefinidamente generando secuencias de símbolos. Por otra parte, si sabemos *a priori* que la teoría **H** es sintácticamente completa, entonces si ocurriese que α no fuese deducible, si que habría de serlo en cambio $\neg\alpha$. Por lo tanto, el método anterior se puede mejorar de manera que termine siempre.

Procedimiento 2 : Búsqueda de una deducción de α o de $\neg\alpha$.

Entrada: Una fórmula α.

Salida: **cierto** si se encuentra una prueba para α en **H**, **falso** si se encuentra una prueba para $\neg\alpha$ en **H**.

```
1: Generar una secuencia finita de símbolos, x.
2: mientras x no es una prueba de α hacer
3:     Generar una nueva secuencia de símbolos y ⟶ x.
4:     si x es una prueba de α. entonces
5:         devolver cierto
6:     fin si
7:     si x es una prueba de ¬α. entonces
8:         devolver falso
9:     fin si
10: fin mientras
11: devolver cierto
```

Puesto que el Procedimiento 2 termina siempre para una fórmula α cualquiera, podemos afirmar lo siguiente.

Resultado 1.
Si una teoría es consistente y sintácticamente completa, entonces es decidible.

El esquema al que nos arroja el Procedimiento 2 se puede resumir en el hecho de que, planteado un problema de decisión, el método nos devuelve siempre una respuesta `cierto/falso`. Si ahora identificamos tales respuestas con 1/0 y reparamos en que la fórmula α puede ser codificada en binario, esto es, como una secuencia de 0's y 1's, es claro que el procedimiento puede ser reformulado en términos de funciones boolenas. Concretamente,

$$\begin{array}{rcl} \psi : \{0,1\}^n & \longrightarrow & \{0,1\} \\ \llcorner \alpha \lrcorner & \longmapsto & \psi(\llcorner \alpha \lrcorner) \end{array} \tag{1.3}$$

Pese al tono halagüeño de esta sección, como consecuencia de los *teroremas de incompletitud* de Gödel, no va ser posible desarrollar estos algoritmos de manera que siempre produzcan una salida. Fueron Church y Turing los que, de manera independiente, repararon en ello.

1.4. MODELOS COMPUTACIONALES.

Es posible que llegados a este punto aún puedan parecernos ambiguos conceptos tales como *algoritmo*, *función computable* o, simplemente, *computar*. No debemos

alarmarnos pues ni si quiera los matemáticos que hemos ido mencionando lograron durante mucho tiempo ponerse de acuerdo. Lo que ocurrió es que tales conceptos se fueron forjando paulatinamente conforme se iban desarrollando los acontecimientos. Más aun, numerosos matemáticos fueron proponiendo alternativas radicalmente diferentes a fin de concretar tales conceptos. Como ya hemos visto, en virtud del Resultado 1, si una teoría es consistente y sintácticamente completa, entonces existirá un procedimiento que permita resolver los distintos problemas de decisión que puedan plantearse. Por otra parte, como ya hemos dejado entrever, Gödel truncó la esperanza por hallar una teoría completa y consistente, pero esto no significaba que necesariamente no pudiese existir un procedimiento de decisión. Por ello, los matemáticos siguieron trabajando en el *Entscheidungsproblem* pero, puesto que los problemas de decisión van ligados a la idea de *algoritmo*, era preciso concretar qué habría de entenderse por tal. De esta manera, surgieron los distintos modelos computacionales como cada una de las distintas alternativas propuestas, las cuales a la postre resultarían ser equivalentes, dando lugar a lo que se conoce como la *tesis de Church-Turing*.

Como hemos insinuado, el fin último de un modelo computacional es esencialmente el de caracterizar las nociones de *algoritmo* y *computación*. Algunos intentos apuntaban en la dirección que introdujimos en la Sección 1.1, esto es, plantearse qué se entiende cuando se computa el valor de una determinada función $f : \mathfrak{A} \longrightarrow \mathfrak{B}$. Este proyecto condujo a la *teoría de funciones recursivas* por parte de Kleene [5].

Se puede decir que una función es *recursiva* cuando, o bien se trata de una función inicial, o bien es una composición de funciones que respeta una serie de reglas de construcción. Las *funciones inciales* propuestas son:

- $\zeta(n) = 0, \forall n \in \mathbb{N}$;
- $\sigma(n) = n + 1, \forall n \in \mathbb{N}$;
- $\pi_i^k(n_1, \ldots, n_k) = n_i, \forall (n_1, \ldots, n_k) \in \mathbb{N}^k$.

Por su parte, las *reglas de construcción* admitidas son:

- *Composición.* Dadas sendas funciones $g : \mathbb{N}^m \longrightarrow \mathbb{N}$ y $h : \mathbb{N}^k \longrightarrow \mathbb{N}$, se define la composición de ambas como $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ según

$$f(n_1, \ldots, n_k) := g(h_1(n_1, \ldots, n_k), \ldots, h_m(n_1, \ldots, n_k)).$$

- *Recursión primitiva.* Dadas sendas funciones $g : \mathbb{N}^k \longrightarrow \mathbb{N}$ y $h : \mathbb{N}^{k+2} \longrightarrow \mathbb{N}$, se define la recursión primitiva, para $m \geq 0$, de ambas como $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ según

$$f(n_1, \ldots, n_k, 0) := g(n_1, \ldots, n_k);$$

$$f(n_1, \ldots, n_k, m + 1) := h(n_1, \ldots, n_k, m, f(n_1, \ldots, n_k, m)).$$

[5] Realmente, la teoría de funciones recursivas es iniciada por Gödel en la demostración de su segundo teorema de incompletitud. Sin embargo, su definición adolecía de ciertas patologías de cara a utilizarse para fundamentar la noción de función computable, debido a que existían funciones que debían ser computables pero que no resultaban ser recursivas. Kleen eliminó esta deficiencia del modelo de Gödel incluyendo el operador μ.

- *Operador μ.* Dada una función $g : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$, se define la acción del operador μ sobre ella como $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ dada por

$$\begin{aligned} f(n_1,\ldots,n_k) &:= \mu x g(n_1,\ldots,n_k,x) \\ &= \min\{x \in \mathbb{N} : g(n_1,\ldots,n_k,x) = 0 \wedge \exists g(n_1,\ldots,n_k,j), \text{ para cada } j = 0,\ldots,x\}. \end{aligned}$$

De esta manera, llegamos a nuestro primer modelo computacional.

Definición 5. *(Modelo computacional de Kleene)*
El modelo computacional de Kleene propone la siguiente caracterización:

- *Una* función computable *es una función recursiva;*
- computar *consiste en calcular el valor de una función recursiva;*
- *un* algoritmo *es una construcción de una función recursiva.*

Por otra parte, un joven matemático llamado Jacques Herbrand reparó, en 1931, en que las funciones $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ podían ser definidas utilizando sistemas de ecuaciones. Gödel desarrolló su idea dando lugar a la *teoría de funciones generales recursivas.* Sin entrar demasiado en detalles, se dice que una función $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ es *generalmente recursiva* si existe un sistema de ecuaciones en forma estándar [6] que garantiza que f está definida en todo su dominio, al que denotamos por $\varepsilon(f)$. Esta definición nos aboca a otro modelo computacional.

Definición 6. *(Modelo computacional de Gödel-Herbrand)*
El modelo computacional de Gödel-Herbrand propone la siguiente caracterización:

- *Una* función computable *es una función general recursiva;*
- computar *consiste en sustituir todas las ocurrencias de cada variable que aparezca en $\varepsilon(f)$ por el mismo número natural y/o sustituir la ocurrencia de cada función que aparezca en $\varepsilon(f)$ por su valor;*
- *un* algoritmo *es un sistema de ecuaciones $\varepsilon(f)$ para una cierta f.*

En torno a los años 1931 y 1933, el matemático y lógico americano Alonzo Church había desarrollado ya otro modelo computacional al que denominó λ-cálculo. De hecho,

[6] Se dice que un sistema de ecuaciones está en forma *estándar* si todas sus ecuaciones son de la forma $f(g_i(\bullet),\ldots,g_j(\bullet)) = \ldots$

el modelo de Kleene fue introducido para probar que el λ-cálculo y las funciones recursivas coinciden. Dada una función f de $n \in \mathbb{N}$ argumentos, se permite que cada unos de estos sea o un número u otra función con sus propios argumentos. De esta manera, lo que ocurre es que se anidan unas funciones en otras y así es posible describir la propia función f juntos con sus argumentos como una secuencia de símbolos que implícitamente representan el valor de $f(\bullet,\ldots,\bullet)$. Las expresiones aceptadas (bien formadas) por el λ-cálculo se denominan λ-términos se definen recurrentemente como

- *Átomo.* Una variable es un λ-término.
- *Abstracción.* Si F es un λ-término y x una variable, entonces $(\lambda x.F)$ es un λ-término.
- *Aplicación.* Si F y G son λ-términos, entonces (FG) es un λ-término.

Esto nos conduce al modelo computacional de Church.

Definición 7. *(Modelo computacional de Church)*
El modelo computacional de Church propone la siguiente caracterización:

- *Una* función computable *es una función λ-definible;*
- computar *es transformar un λ-término inicial en otro final;*
- *un* algoritmo *es un λ-término.*

Otros matemáticos se inspiraron en la manera en que los seres humanos nos comunicamos, esto es, en los lenguajes. La idea radica en que si bien los seres humanos recibimos una secuencia de palabras para después producir otra, la cual debe haber sido procesada de acuerdo a unas determinadas reglas (nótese que estamos pensando en seres humanos con capacidad cognoscitiva plena y sin ningún tipo de trastorno psíquico). En el fondo, esto puede interpretarse como que la secuencia que recibimos constituye la descripción misma de un problema, mientras que la que producimos sería la propuesta de solución al propio problema.

En 1951, un matemático ruso proveniente de afamada estirpe de matemáticos llamado Andrey Markov Jr. describió un modelo computacional que hoy día conocemos como *gramáticas*. Este consiste en una secuencia finita de *producciones*, digamos M, tales como

$$\begin{aligned} a_1 &\longrightarrow b_1 \\ &\vdots \\ a_k &\longrightarrow b_k \end{aligned}$$

donde $a_i, b_i, i = 1,\ldots,k$, son palabras sobre un determinado alfabeto Σ.

Sea M una gramática de Markov y $n \in \mathbb{N}$ un cierto numero natural fijo. Podemos definir una función, f_M^n como $f_M^n(a_1, \ldots, a_n) := b$, siendo b la palabra generada efectivamente por la secuencia de producciones M. Señalemos que nada nos impide codificar las palabras de entrada y salida mediante números. De esta manera, se dice que una función arbitraria, f, es Markov-computable si existe una gramática M tal que $f \equiv f_M^n$.

Definición 8. *(Modelo computacional de Markov)*
El modelo computacional de Markov propone la siguiente caracterización:

- *Una* función computable *es una función Markov-computable;*
- computar *es ejecutar una gramática;*
- *un* algoritmo *es una gramática.*

Demos paso al que sin duda es el modelo computacional más conocido. Su fama se debe a que se basa en una idea completamente diferente al resto de los modelos que hemos presentado hasta ahora. En su artículo de 1936, Alan Turing es capaz de abstraer la actividad mecánica que realiza un ser humano cuando resuelve un problema, descomponiéndola a su vez en varias acciones canónicas. Tal actividad puede ser recreada por la denominada *máquina de Turing* [7].

La máquina de Turing consta de varios componentes. A saber,

- una *unidad de control*, en clara alusión al cerebro humano;
- una *cinta* de longitud potencialmente infinita dividida en *celdas* del mismo tamaño, correspondiente al papel que utilizaría un humano al realizar sus cálculos;
- un *lector* que se puede mover por cada una de las celdas y que es capaz de escribir, borrar y reconocer los caracteres que aparezcan en estas, simulando lo que haría el ojo y la mano, provista de un lápiz, de un humano.

De esta manera, la unidad de control siempre se encuentra en un cierto *estado* de entre todos los posibles estados en los que puede estar, los cuales constituyen un conjunto finito. Dos de estos estados son los que se denominan *inicial* y *final*. También se ha de contar con un *programa* dado por una serie finita de instrucciones incluidas en la unidad de control al que se denomina *programa de Turing*. Se dice que dos máquinas de Turing son distintas cuando tienen implementados distintos programas de Turing.

[7] Reparemos en que al utilizar el término *máquina*, Turing pretende reflejar que desde el punto de vista del formalismo de Hilbert, no existe nada en la acción de *calcular* que no pueda ser implementado por un dispositivo mecánico. Además, conviene señalar que lo utiliza con anterioridad a la aparición de los primeros ordenadores. De hecho, serán sus propios trabajos los que acaben por constituir una de las piedras angulares de la *informática*.

Antes de ejecutar una máquina de Turing, esta debe encontrarse en el estado inicial y preparada para leer la llamada *palabra de entrada* de un cierto alfabeto Σ que ha sido escrita con anterioridad en una celda de la cinta. Desde su ejecución, una máquina de Turing va realizando de forma mecánica una serie de operaciones según indique el programa. En cada etapa, la máquina lee el símbolo de la celda sobre la que se encuentre el lector y, atendiendo al estado en el que se encuentre, realizará una de las siguientes acciones:

- borrar el símbolo y reemplazarlo por otro;
- mover el lector de la celda actual a una de las adyacentes;
- modificar el estado de la unidad de control.

La máquina se detiene si la unidad de control alcanza el estado final o bien, si no restan instrucciones que realizar. Las palabras de entrada y de salida (si es que la máquina se detiene) pueden codificarse mediante sendas secuencias binarias, lo cual posibilita su representación vía funciones booleanas para así concebir que la máquina trabaja únicamente con valores numéricos. Identificando las palabras $a_1, \ldots, a_k, b \in \Sigma$, siendo Σ un determinado alfabeto, con $\llcorner a_1 \lrcorner, \ldots, \llcorner a_k \lrcorner, \llcorner b \lrcorner$, podemos concebir la acción de una máquina de Turing, T, como una función $f : \Sigma^* \longrightarrow \Sigma^*$ dada por $f_T^k(a_1, \ldots, a_k) := b$. Así, dada una función, f, cualquiera, se dice que esta es Turing-computable si existe alguna máquina de Turing tal que $f \equiv f_T^k$.

Definición 9. *(Modelo computacional de Turing)*
El modelo computacional de Turing propone la siguiente caracterización:

- *Una* función computable *es una función Turing-computable;*
- computar *es ejecutar programa de Turing en una maquina de Turing;*
- *un* algoritmo *es un programa de Turing.*

Restan aún otros modelos computacionales como el de Post o el de Kolmogórov-Uspenski, pero hemos decidido no incorporarlos pues realmente no aportan demasiado toda vez que ya se han presentado suficientes modelos. En cambio, si que se ha incorporado un Apéndice a este trabajo que pretender poner de manifiesto la mayor complejidad de otros modelos computacionales, los cuales resultan laberínticos incluso para el iniciado en matemáticas, frente al modelo de Turing.

1.5. LA MÁQUINA DE TURING.

La máquina de Turing fue concebida inicialmente con objeto de formalizar los conceptos de *algoritmo, computación* y *función computable.* El modelo de Turing original

se presentó en su artículo [17], si bien es cierto que él mismo introduciría más tarde diversas variantes que no eran más que generalizaciones del primero. Además, Turing terminaría por demostrar que tales variante no surtían ningún efecto en cuanto al aumento de la capacidad computacional del modelo primigenio.

La máquina de Turing original cuenta con una *unidad de control* que incorpora el *programa de Turing*, una *cinta* dividida en *celdas* idénticas, y un *lector* móvil dispuesto sobre la cinta y conectado a la unidad de control. Asimismo, se imponen los siguientes requisitos:

- La *cinta* se utiliza para escribir y leer las palabras iniciales, intermedias y finales. Además, ha de ser potencialmente infinita en un sentido. En cada *celda* debe aparecer un símbolo perteneciente a un alfabeto, el *alfabeto de la máquina*, $\Delta = \{z_1, z_2, z_3 \ldots, z_t, \sqcup\}$, con $t \geq 3$. Se suele requerir que $z_1 = 0, z_2 = 1$, así como que $z_t = \sqcup$. Al símbolo $\sqcup$ se lo conoce como *espacio en blanco*. La *palabra de entrada* deber pertenecer a otro alfabeto, Σ, que ha de ser tal que $\{0,1\} \subseteq \Sigma \subseteq \Delta - \{\sqcup\}$ y al que llamamos *alfabeto de entrada*. Inicialmente, todas las celdas a excepción de aquellas que contienen la palabra de entrada se encuentran vacías, esto es, muestran el símbolo $\sqcup$.

- La *unidad de control* se halla siempre en algún *estado* que pertenece al denominado *espacio de estados* $Q = \{q_1, \ldots, q_s\}$. Al estado q_1 se lo conoce como *estado inicial*. Por su parte, se dice que algunos de los estados son *finales*. Al conjunto de estados finales se lo denota por $F \subseteq Q$. A los estados pertenecientes a $Q - F$ se los llama *estados no-finales*. Cuanto no resulte relevante el índice del estado, utilizaremos q_{si} y q_{no} para referir a un estado final y a uno no-final cualesquiera, respectivamente.

- La unidad de control incorpora el *programa de Turing*. El programa es capaz de dirigir las distintas componentes de la máquina y es lo que caracteriza a cada una de las máquinas de Turing. Formalmente, un programa de Turing es una función parcial $\delta : Q \times \Delta \longrightarrow Q \times \Delta \times \{\text{Izquierda, Derecha, Permanecer}\}$. Se asume que $\delta(q_{no}, z) \downarrow$ para algunos $z \in \Delta$ y que $\delta(q_{si}, z) \uparrow$ para todo $z \in \Delta$, esto es, que siempre existe una transición desde un estado no-final y ninguna desde un estado final.

- El lector puede moverse a cada una de las celdas de la cinta, leer sus símbolos y modificar el contenido de las mismas.

- Antes de ejecutar una máquina de Turing debe haber acontecido:
  - que la palabra de entrada haya sido escrita al comienzo de la cinta;
  - que el lector se haya situado en la primera celda de la cinta;
  - que la unidad de control se encuentre en el estado inicial.

- La máquina de Turing actúa de manera independiente, paso a paso y conforme al programa de Turing especificado por δ. En concreto, si la máquina de Turing se encuentra en el estado $q_i \in Q$ y lee el símbolo $z_k \in \Delta$, entonces:
  - se detiene, si q_i es un estado final;

- se detiene, si $\delta(q_i, z_k) \uparrow$;
- pasa al estado q_j, escribe z_r y, o mueve el lector a derecha o izquierda, o bien permanece en el mismo sitio, si $\delta(q_i, z_k) = (q_j, z_r, D)$.

El propio Turing presentó determinadas modificaciones sobre el modelo original. De hecho, en el planteamiento primigenio se contemplaba el uso de una única cinta infinita en ambos sentidos y, por tanto, lo que nosotros hemos presentado supone en si mismo una variación del modelo original. Otras variantes permiten el uso de diversas cintas e, incluso, de cintas multidimensionales. Sin embargo, la capacidad computacional de todas ellas resulta ser la misma (ver [14]). Demos paso a la definición formal de máquina de Turing más aceptada actualmente.

Definición 10. *(Definición formal de la máquina de Turing)*
Se define formalmente la máquina de Turing *como una* $7-$*tupla*

$$T = (Q, \Sigma, \Delta, \delta, q_1, \sqcup, F),$$

donde $Q, \Sigma, \Delta, \delta, q_1, \sqcup, F$ *son los objetos que hemos introducido anteriormente.*

Ahora que ya contamos con una definición precisa de la máquina de Turing, cabe preguntarse al respecto de la manera de representar las distintas etapas o pasos que tienen lugar durante la ejecución de la misma. Para ello, se recurre al concepto de *configuración interna* de la máquina.

Definición 11. *(Configuración interna de la máquina de Turing)*
Sea T *una determinada máquina de Turing. Sea también* w *una cierta palabra de entrada para* T. *La* configuración interna *de* T *después de una cantidad finita de etapas es la palabra* uq_iv *tal que*

- q_i *es el estado actual;*
- $uv \in \Delta^*$ *está formada por (1) los símbolos que no son espacios en blanco de la cinta que se encuentran más a la derecha o, en su caso, (2) por el símbolo que se encuentra inmediatamente a la izquierda de la cinta y que es el que se encuentra más a la derecha. Se asume que* $v \neq \lambda$ *si (1) y que* $v = \lambda$ *si (2);*
- T *lee el símbolo más a la izquierda de* v *si (1), o el símbolo* $\sqcup$ *si (2).*

El modelo de la máquina de Turing que hemos presentado se caracteriza por tener determinada la acción que realizar a cada etapa de manera unívoca. Por ello, a este modelo se lo suele conocer como *máquina de Turing determinista*. Por otra parte,

fueron concebidas otras máquinas que, a diferencia de las deterministas, son capaces de realizar diferentes operaciones en un mismo paso, dando lugar a los consiguientes resultados distintos. Es decir, mientras que el resultado de una máquina determinista queda determinado por la palabra de entrada que se introduzca, una máquina no determinista puede producir distintos resultados a partir de una misma entrada. A este tipo de máquinas no deterministas se las conoce como *máquinas indeterministas.* La introducción del intederminismo resulta ser una herramienta de suma transcendencia debido a que aumenta la potencia computacional del modelo original al permitir resolver de manera inmediata problemas que antes eran altamente costosos. Es un problema sustancial el dirimir si tal aumento de la potencia computacional es auténtico en el sentido de si realmente resuelve problemas que no podían ser resueltos por máquinas deterministas. A esta cuestión se la conoce como *conjetura de Cook* y volveremos sobre ella más adelante.

El programa de una máquina de Turing indeterminista se caracteriza porque sus instrucciones vienen dadas por una función

$$\delta_I : Q \times \Delta \longrightarrow \left(Q \times \Delta \times \{\text{Izquierda, Derecha, Permanecer}\}\right)^p, \tag{1.4}$$

dada por

$$\delta_I(q_i, z_k) = \Big((q_{j_1}, z_{r_1}, D_1), \ldots, (q_{j_p}, z_{r_p}, D_p)\Big). \tag{1.5}$$

Existen dos alternativas a la hora de concebir la acción de una máquina de Turing indeterminista. La primera de ellas es la que imagina que la máquina es capaz de elegir la opción más conveniente de entre las distintas posibilidades de manera que minimiza el número de etapas requeridas para resolver un problema. Por su parte, la otra figura que la máquina de Turing se ramifica dando lugar a "submáquinas" las cuales llevan a cabo cada una de las distintas alternativas.

Hasta ahora hemos permitido que la máquina de Turing trabaje con un determinado alfabeto Δ compuesto por un número arbitrario de elementos. Sin embargo, las cosas se simplifican notablemente cuando se trabaja con alfabetos de la máquina y de entrada de tamaño reducido. La elección más sencilla resulta ser $\Delta = \{0, 1, \sqcup\}$ y $\Sigma = \{0, 1\}$. Esta manera de proceder no desemboca en una pérdida de generalidad pues cualesquiera que sean los alfabetos originales, estos pueden ser codificados en código binario. Es posible encontrar una demostración de esta equivalencia en [14]. Además, se suele exigir que el conjunto de estados finales conste de un único elemento, esto es, $F = \{q_F\}$. Cuando se lleva a cabo esta simplificación se habla del *modelo de Turing reducido* y de *máquinas de Turing reducidas.* Así, formalmente se dice que una máquina de Turing reducida es una $7-$tupla

$$T = (Q, \{0, 1\}, \{0, 1, \sqcup\}, \delta, q_1, \sqcup, \{q_F\}). \tag{1.6}$$

Por otra parte, a menudo es interesante el disponer de manera clara de una representación de una máquina de Turing concreta. Esta ha de ser tal que permita recuperar toda la información relativa a la propia máquina y, por ello, el procedimiento

adecuado resulta ser el de codificar y, en particular, hacerlo en binario. En efecto, lo que se pretende es codificar la función de transición, δ, pues resulta ser el objeto determinante y característico de cada máquina. Así, si $\delta(q_i, z_j) = \left(q_k, z_l, D_m\right)$ es una instrucción del programa de Turing de una cierta máquina T, podemos codificarla como

$$I = 0^{(i)}10^{(j)}10^{(k)}10^{(l)}10^{(m)}, \tag{1.7}$$

donde $D_1 =$ Izquierda, $D_2 =$ Derecha y $D_3 =$ Permanecer.

A partir de las distintas codificaciones de cada una de las instrucciones, se puede obtener el *código de la máquina* tras una especie de concatenación llevada a cabo de la siguiente manera: si $I_1, \ldots, I_n$ son las codificaciones de cada una de las n instrucciones, respectivamente, se obtiene el código de la máquina como

$$\llcorner T \lrcorner = 111I_1 11I_2 11 \ldots 11I_n 111. \tag{1.8}$$

Por su parte, resulta claro que existe una biyección entre los números naturales y sus representaciones en binario. De esta manera, es posible recuperar de manera unívoca el número natural (su expresión decimal) al que representa $\llcorner T \lrcorner$. A tal número natural se lo conoce como *índice* de la máquina T. Debemos reparar en que algunos números naturales no son índices de ninguna máquina de Turing, concretamente, aquellos que no tienen la forma que hemos presentado antes. Como consecuencia de ello, se suele aceptar que tales números naturales que no representan máquinas de Turing resultan ser el índice de una máquina concreta denominada *máquina de Turing vacía* y que se detiene para cualquier palabra de entrada en 0 etapas. Tras aceptar este convenio, podemos afirmar el siguiente resultado.

Resultado 2.
Todo número natural es el índice de una máquina de Turing y solo de una.

En 1936, Alan Turing reparó en que era posible concebir una máquina de Turing que fuese capaz de imitar la acción de cualquier otra máquina de Turing. Esta máquina es la que se conoce como *máquina de Turing universal* e incorpora, no una, sino tres cintas divididas en celdas homogéneas. La primera de ella, la *cinta de entrada* contiene la palabra de entrada la cual, a su vez, está compuesta por el índice de una máquina de Turing cualquiera y una cierta palabra arbitraria. La segunda es la *cinta de trabajo*, la cual inicialmente se encuentra vacía y se utiliza de manera similar a como lo haría una máquina de Turing básica. La tercera se conoce como *cinta auxiliar* y sirve para grabar el estado actual en el que se encuentra la máquina que se está simulando e ir comprobando si este es un estado final de la misma. Debemos ser conscientes de que, en realidad, la máquina de Turing universal no es de una naturaleza distinta a una máquina estándar. Esto supone que el siguiente resultado sea de enorme trascendencia.

Resultado 3.
Existe una máquina de Turing capaz de computar todo aquello que resulta computable por alguna máquina de Turing.

La existencia de la máquina de Turing universal resulta ser verdaderamente relevante pues pone de manifiesto que es factible diseñar un método suficientemente general como para resolver cualquier problema que pueda resolverse. Notemos también que, puesto que la máquina de Turing universal utiliza como datos de entrada el índice de la máquina que va a ser simulada junto con la propia palabra de entrada, no ha de existir en principio diferencia alguna entre lo que son meros datos y lo que son instrucciones. El propio Turing reparó en este hecho estableciendo que la distinción entre ambos viene fijada por la interpretación que se realice. Por otra parte, la prueba al respecto de la existencia de la máquina de Turing universal parecía respaldar la clarividencia que había mostrado un siglo atrás el matemático británico Charles Babbage, quien postuló que sería posible construir una máquina física que fuese capaz de resolver mecánicamente cualquier problema computable.

1.6. FUNCIONES COMPUTABLES Y LENGUAJES.

El modelo de Turing puede ser empleado para computar valores de una determinada función, para generar elementos de un conjunto y para dilucidar si un determinado elemento pertenece a un determinado conjunto o no, entre otras cosas. El último de los casos es que va a acaparar nuestra atención debido a que de manera natural en el contexto que nos ocupa uno ha de interesarse por conocer cuándo una determinada proposición es o no un elemento de un conjunto que es la *teoría*, es decir, cuando es o no un *teorema*.

A continuación vamos a presentar diversas definiciones elementales, así como formalizaremos la noción de *función computable* relativa al modelo de Turing.

Definición 12. *(Resultado, conjunto de parada)*
Sea $T = (Q, \Sigma, \Delta, \delta, q_1, \sqcup, F)$ *una determinada máquina de Turing y sea* $w \in \Sigma^*$ *una cierta palabra de entrada. Se denomina* resultado *de* T *y a la función*

$$Res_T : \Sigma^* \longrightarrow \{Aceptar,\ Rechazar,\ No\ reconocido\}$$

que viene dada por

$$Res_T(w) = \begin{cases} Aceptar & si \quad T \text{ se detiene en un estado final } q_{si}; \\ Rechazar & si \quad T \text{ se detiene en un estado no-final } q_{no}; \\ No\ reconocido & si \quad T \text{ no se detiene.} \end{cases}$$

Por su parte, al conjunto

$$\mathbb{P}_T = \{w \in \Sigma^* : Res_T(w) \in \{Aceptar, Rechazar\}\}$$

se lo conoce como conjunto de parada *de la maquina* T.

Las nociones previas nos conducen inexorablemente a la de *palabra aceptada* por una máquina de Turing, así como la de *lenguaje aceptado* por ella. Formalicemos estas nociones.

Definición 13. *(Palabra aceptada, lenguaje aceptado)*
Sea $T = (Q, \Sigma, \Delta, \delta, q_1, \sqcup, F)$ *una determinada máquina de Turing. Se dice que una palabra* $w \in \Sigma^*$ *es* aceptada *por* T *si, y solo si,* $Res_T(w) = Aceptar$. *Asimismo, a la contraimagen por la función resultado de Aceptar se lo denomina* lenguaje aceptado *por* T *o también,* lenguaje propio *de* T, *esto es,*

$$\mathbb{L}_T = Res_T^{-1}(\{Aceptar\}) = \{w \in \Sigma^* : Res_T(w) = Aceptar\}.$$

Al fin estamos en condiciones de formalizar rigurosamente los conceptos de *función computable* y *problema de decisión*. Sin embargo, en aras de la claridad de claridad expositiva, vamos a ceñirnos desde ahora al caso reducido, esto es, $\Sigma = \{0, 1\}$. Como ya mencionamos, esto no supone una pérdida de generalidad pues cualquier alfabeto puede ser codificado sobre Σ. Además, identificaremos los elementos Aceptar, Rechazar del conjunto de llegada de las funciones resultado con $\{0, 1\}$, concretamente, Aceptar $\equiv 1$ y Rechazar $\equiv 0$. De esta manera, podría decirse que una función resultado es tal que $\text{Res}_T : \Sigma^* \longrightarrow \Sigma \cup \{\text{No reconocido}\}$. Con esto en mente, demos paso a la siguiente defineción.

Definición 14. *(Función computable)*
Sea ψ *una función booleana. Se dice que* ψ *es computable si existe una máquina de Turing,* T, *de manera que*

$$\psi(w) = Res_T(w) \quad \forall w \in \Sigma^*.$$

Desde otra perspectiva, resulta claro que, dada una cierta función booleana, ψ, podemos plantearnos la determinación del lenguaje

$$\mathbb{L}_\psi = \{w \in \Sigma^* : \psi(w) = 1\}. \tag{1.9}$$

Tal determinación efectiva es lo que se conoce como *problema de decisión* y será factible si la función ψ es computable y, en tal caso, se dice que el problema de decisión es *decidible*

o *computable*. En otro caso, se manifiesta que es *indecidible* o *incomputable*. Dada la estrecha relación entre las funciones booleanas y sus lenguajes asociados como queda patente en (1.9), un problema de decisión puede ser entendido como, dado un lenguaje $\mathbb{L} \subseteq \Sigma^*$, encontrar una máquina de Turing, T, con $\mathbb{P}_T = \Sigma^*$ tal que

$$\mathrm{Res}_T(w) = \chi_{\mathbb{L}}(w) \quad \forall w \in \Sigma^*, \tag{1.10}$$

donde $\chi_{\mathbb{L}}$ es la *función característica* del lenguaje, esto es, $\chi_{\mathbb{L}} : \Sigma^* \longrightarrow \Sigma$ tal que

$$\chi_{\mathbb{L}}(w) = \begin{cases} 1 \quad si \quad Tw \in \mathbb{L}; \\ \\ 0 \quad si \quad w \notin \mathbb{L}. \end{cases} \tag{1.11}$$

Reparemos en que en el caso que nos ocupa, esto es, $\Sigma = \{0,1\}$, las funciones características resultan ser también funciones boolenas. De esta manera, damos paso a las consiguientes definiciones de computabilidad relativas a lenguajes.

Definición 15. *(Lenguaje decidible, semi-decidible, indecidible)*
Se dice que un lenguaje $\mathbb{L} \subseteq \Sigma^*$ *es* semi-decidible *o* recursivamente enumerable *si existe una máquina de Turing* T *tal que*

$$\mathbb{L} = \mathbb{L}_T.$$

Si un lenguaje semi-decidible $\mathbb{L} \subseteq \Sigma^*$ *es tal que* $\Sigma^* - \mathbb{L}$ *también es semi-decidible, entonces se dice que es* decidible, recursivo *o* computable. *Los lenguajes no decidibles se conocen como* indecidibles.

Una vez que hemos establecido todas estas nociones es sensato preguntar que problemas pueden ser computados. Si atendemos a la implícita conexión entre computabilidad y resolución efectiva de un problema, nuestra intuición nos anima a contestar que no todos los problemas han de ser necesariamente computables. En efecto, el propio Alan Turing demostró en [17] que existe un problema, el conocido *problema de la parada*, que no lo es. Asimismo, existen otras demostraciones que utilizan procedimientos diagonales para poner de manifiesto la existencia de problemas indecidibles. Nosotros finalizaremos este capítulo presentando un argumento que solo requiere pequeñas nociones al respecto de la cardinalidad de un conjunto.

Resultado 4.
Existen infinitos problemas de decisión no computables.

Demostración. Atendiendo a la Definición 1, resulta claro que se puede establecer fácilmente, utilizando la longitud, un morfismo de semigrupos entre $\langle \mathscr{L}^*(\Sigma), \bullet \rangle$ y $\langle \mathbb{N}, + \rangle$. Así, aunque Σ es finito (es un alfabeto), el conjunto de todas las palabras de cualquier

longitud, $\mathscr{L}^*(\Sigma)$, es infinito numerable y, por lo tanto, cada lenguaje tendrá a lo sumo cardinalidad numerable. Además, respecto a la cantidad de lenguajes posibles, esto es, $\left|\{\mathbb{L} \subseteq \mathscr{L}^*(\Sigma)\}\right|$, se tiene que

$$\left|\{\mathbb{L} \subseteq \mathscr{L}^*(\Sigma)\}\right| = \left|\mathscr{P}(\mathscr{L}^*(\Sigma))\right| = 2^{\aleph_0}. \tag{1.12}$$

Asimismo, cada lenguaje define un único problema de decisión, por lo tanto, existe una cantidad infinita no numerable de problemas de decisión. Por otra parte, en virtud del Resultado 2, existe tan solo una cantidad numerable de máquinas de Turing y, consiguientemente, no es posible asociar a cada problema una máquina de Turing que lo resuelva. Así, puesto que la cantidad de problemas computables es numerable mientras que la cantidad de problemas de decisión no lo es, existe una cantidad infinita no numerable de problemas indecidibles.

■

1.7. REFERENCIAS BIBLIOGRÁFICAS.

Debemos reparar en que a lo largo de estas páginas hemos tratado aspectos bastante diversos como son la fundamentación de la matemática y la teoría formal de lenguajes. Describiremos en las líneas posteriores los aspectos más reseñables de la bibliografía que hemos utilizado para la elaboración de este trabajo.

- El libro de Robic ([14]) realiza una exposición bastante clara y accesible a casi todos los aspectos que se han cubierto en esta parte, si bien es cierto que algunas cuestiones pueden resultar densas o inaccesibles para el lego en matemáticas. Pormenorizadamente, los capítulos 2, 3 y 4 explican en profundidad el nacimiento de la teoría de la computación. Por su parte, el capítulo sexto expone el modelo de la máquina de Turing. Finalmente, aunque nosotros hemos pasado de soslayo sobre los problemas indecidibles, los capítulos 8 y 9 están dedicados a ahondar en ellos.

- Por su parte, no podemos dejar de mencionar las obras de Kolmogórov y Dragalin ([7]- [8]), las cuales constituyen una vía excepcional para los matemáticos que deseen iniciarse en el estudio de la lógica en general y de la computabilidad en particular.

- Asimismo, el celebre artículo de Alan Turing ([17]), aunque tiene partes más técnicas, incorpora una cuidadosa descripción de los planteamientos primordiales que resulta claramente clarificadora.

- También merecen ser mencionados los apuntes de la asignatura *Lógica e informática*, integrada en el máster que nos atañe, elaborados por el profesor José Pedro Úbeda, los cuales presentan las nociones más específicas con absoluto rigor.

2

UNA INTRODUCCIÓN A LA *teoría de la complejidad.*

"Si la gente no cree que las matemáticas son sencillas,
es solo porque no se da cuenta de lo complicada que es la vida."
— John von Neumann.

Resumen: En primer lugar, se presenta el propósito de la *teoría de la complejidad*. A continuación, se presentan los conceptos más importantes de la misma así como las clases de complejidad más relevantes. Por último, se expone la *conjetura de Cook* y se realizan distintos comentarios al respecto.

2.1. ¿QUÉ ES LA COMPLEJIDAD?

La *complejidad* resulta ser una noción con la que estamos habituados a lidiar en nuestro día a día. Desde nuestra etapa escolar sabemos lo que es que un profesor nos plantee en un examen un problema demasiado difícil, así como lo costoso de aprender a tocar un instrumento musical de forma que las notas encajen de esa manera que han de hacerlo para no perturbar el sosiego de nuestros oídos. También sabemos que es más fácil sumar dos números que multiplicarlos y que, en ambos casos, es preferible y más fiable utilizar la calculadora. Al fin y al cabo, la inmensa mayoría de la población adulta cree firmemente que la ardua e, incluso, cruel tarea de utilizar papel y lápiz resulta ser absolutamente defectible. En efecto, si bien es cierto que es conveniente saber realizar ciertos procedimientos de manera mecánica, no debemos olvidar que esto termina por ser una actividad irreflexiva, una mera acción no deliberada sin pensamiento consciente. Sin embargo, este tipo de conocimiento puede ser de utilidad cuando uno pretende, por ejemplo, conducir un coche a gran velocidad, evitando reflexionar y actuando de manera mecánica para salvaguardar su integridad personal. Pero este planteamiento carece de sentido cuando se lleva al ámbito de lo matemático pues, después de todo, la probabilidad de que un polinomio atente contra nosotros es escasa. Por ello, la gente no se equivoca cuando cree que calcular mentalmente o a mano las operaciones que se van requiriendo es una empresa baldía toda vez que siempre se dispone de una calculadora a mano.

De esta manera, la atención ha de dirigirse hacia otro tipo de cuestiones, concretamente, hacia aquellas para las que no existe un método que permita resolverlas. En un ejercicio de autorreferencialidad, el proponer una clasificación de los problemas según su complejidad resulta ser una de tales cuestiones. En un primer acercamiento al asunto, dispondremos enseguida que tal clasificación ha de respetar nuestra intuición considerando los problemas que cuentan con un método que permite resolverlos como más "fáciles" que aquellos que no lo hacen pues, por muy tedioso que resulte un método, basta con ser paciente y seguir sus instrucciones con recelo para dar con la solución buscada. Sin embargo, también sabemos gracias a la experiencia que existen problemas que se resuelven *pensando* sin que medie ningún tipo de procedimiento. Acaba de entrar en escena lo que de conoce como *indeterminismo*, esto es, nuestra clasificación ha de ser suficientemente general como abarcar todos los problemas sin prestar atención a un método concreto de resolución. En efecto, no estamos interesados en cuantificar el coste de un determinado método, es decir, en medir cuán farragoso es sino en analizar una propiedad de los propios problemas.

Volvamos sobre la dificultad del problema de multiplicar dos números frente a la de sumarlos. Aunque probablemente todo el mundo concedería que, efectivamente, es más complejo realizar una multiplicación, debemos preguntarnos la razón subyacente a esta creencia. Quizás se deba a que aprendemos a sumar antes de multiplicar y por ende asimilamos que el saber sumar es un requisito para poder multiplicar. Pero, aunque razonemos de esta manera, seguimos sin poder demostrar que multiplicar es más complicado. Cualquiera podría aducir que multiplicar lleva más tiempo que sumar, pero esto se debe a que está pensando en el método tradicional de multiplicación. Esto, en primer lugar, resulta ser un planteamiento ingenuo pues manifiesta que se confunde lo que

es propiamente un problema de lo que es un método para resolverlo. Además, hoy en día sabemos, por ejemplo, que existen algoritmos que multiplican números grandes mucho más deprisa que el método tradicional. Así que la pregunta que debemos plantearnos para catalogar la dificultad de los problemas es si todos y cada uno de los métodos que pueden ser utilizados para realizar multiplicaciones requieren de más tiempo que el mejor de los que se utilizan para sumar. Para responder a esta cuestión se recurre habitualmente a un argumento de imposibilidad, es decir, que no es posible encontrar un algoritmo que realice multiplicaciones más eficiente que el utilizado para sumar. De este tipo de cuestiones se ocupa la llamada *teoría de la complejidad.*

2.2. ALGUNAS NOCIONES DE LA COMPLEJIDAD.

Como hemos anunciado en la sección previa, la teoría de la complejidad se ocupa de proponer una taxonomía para los problemas conforme a la eficiencia con la que pueden ser resueltos. La eficiencia de un método se suele medir a partir de los recursos que han de utilizarse para llevar a cabo una tarea. En el ámbito de la computación los recursos atañen a dos aspectos: el espacio y el tiempo. El primero de ellos está relacionado con la cantidad de memoria de almacenamiento requerida por un método. Podemos pensar que, utilizando lápiz y papel, necesitamos de varios folios para solucionar un problema, mientras que el tiempo simplemente refleja la duración del proceso de cómputo. Si se nos provee además de una goma de borrar, podremos reutilizar los folios de los que disponíamos, optimizando el coste de espacio. Sin embargo, no disponemos de una suerte de goma para el tiempo y, por ello, consideramos el coste temporal más importante que el espacial. Por ello, el recurso que se suele estudiar habitualmente es el tiempo y así lo haremos nosotros.

En primer lugar, esperamos que el lector convenga con nosotros a partir de lo expuesto en la Sección 1.4 que el modelo computacional más manejable es el de la máquina de Turing. Por ello, será el que utilicemos como marco común de referencia para la catalogación de los problemas. Asimismo, toda máquina de Turing no trivial comienza por leer la palabra de entrada antes de proseguir con su computación. Por ello, resulta claro que el coste temporal debe depender de la palabra de entrada que se contemple en cada caso. Esto nos conduce a la siguiente definición.

Definición 16. *(Tiempo de cómputo para una palabra)*
Sea T una máquina de Turing con alfabeto de entrada Σ y sea $w \in \mathbb{P}_T$ una cierta palabra de entrada. Se define el tiempo de cómputo *de T para w como la función $t_T : \mathbb{P}_T \longrightarrow \mathbb{N}$, que asigna a cada w en número de pasos requeridos por T hasta que se detiene.*

La definición anterior solo alude a una palabra de entrada concreta y, además, estamos de acuerdo en que la longitud de la palabra de entrada influye sobre el tiempo de cómputo

de una máquina. Asimismo, usualmente se sigue una doctrina *ad cautelam* y, por tanto, la complejidad de una máquina estará relacionada con el caso más desfavorable de todos los posibles. Esto nos lleva a la definición siguiente.

Definición 17. *(Función de complejidad temporal)*
Sea T una máquina de Turing con alfabeto de entrada Σ y sea $w \in \mathbb{P}_T$ una cierta palabra de entrada. Se define la función de complejidad temporal *como*

$$T_T : \mathbb{N} \longrightarrow \mathbb{N}$$

dada por

$$T_T(n) = \max_{|w| \leq n} \{t_T(w)\}.$$

En algunas ocasiones, considerar el caso más desfavorable puede distorsionar la realidad del problema. En consecuencia, ocasionalmente se prefiere trabajar con la complejidad media. Nosotros no trataremos esta noción. Vamos a presentar ahora otro de los conceptos propios de la teoría de la complejidad.

Definición 18. *(Función temporalmente computable)*
Se dice que una función $f : \mathbb{N} \longrightarrow \mathbb{N}$ es temporalmente computable *si existe una máquina de Turing, T, con alfabeto $\Sigma = \{1\}$ que la computa de manera que $\mathbb{P}_T = \Sigma^*$, $f(n) \geq n$ y $T_T(n) = O(f(n))$.*

Cabe realizar varios comentarios al respecto de la definición anterior. En primer lugar, la elección del alfabeto $\Sigma = \{1\}$ supone que $\Sigma^* = \mathbb{N}$ sin más que realizar la identificación $1^{(n)} = n$ [1]. Por otra parte, la condición $f(n) \geq n$ garantiza que la máquina de Turing considerada es capaz de, al menos, leer la palabra de entrada al completo. En cuanto a $T_T(n) = O(f(n))$, cabe señalar que lo que quiere decir es que T_T está acotada superiormente por f cuando $n \longrightarrow \infty$, esto es, que T_T es inferior a f para valores de n suficientemente grandes [2]. Finalmente, conviene indicar que, entre otras, las funciones polinómicas resultan ser temporalmente computables.

Por otra parte, en el Capítulo 1 señalábamos la robustez del modelo de Turing en el sentido de que la capacidad computacional del mismo no se ve alterada por el alfabeto que utilice cada máquina pues era suficiente realizar las codificaciones pertinentes. Por supuesto que esto se mantiene en el ámbito de la teoría de la complejidad y, por lo tanto, no ahondaremos en ello. Se puede encontrar una demostración en [10].

[1] Esta identificación es habitual cuando se trabaja con máquinas de Turing como puede comprobarse en el Apéndice.
[2] A esta notación se la conoce como *notación de Landau*.

2.3. CLASES DE COMPLEJIDAD.

Una vez que ya hemos introducido todas las herramientas que precisamos, estamos en disposición de presentar, al menos, las clases de complejidad más importantes. Se dice que una *clase de complejidad* es un conjunto de problemas de decisión que pueden ser computados, según el modelo de Turing, utilizando una cantidad de recursos delimitada por una misma cota. La primera clase que presentaremos serán las que solo permiten máquinas de Turing deterministas.

Definición 19. *(Clases **DTIME**)*
Dada una función temporalmente computable $f : \mathbb{N} \longrightarrow \mathbb{N}$. *Se define la clase* $\boldsymbol{DTIME}(f)$ *como*

$$\boldsymbol{DTIME}(f) = \left\{\mathbb{L} \subseteq \Sigma^* : \exists T \text{ con } Res_T = \chi_{\mathbb{L}} \wedge T_T \in O(f)\right\},$$

donde T *es una máquina de Turing determinista.*

Si se permite que la máquina de Turing sea indeterminista, entonces de forma similar se definen las clases de complejidad **NTIME**.

Definición 20. *(Clases **NTIME**)*
Dada una función temporalmente computable $f : \mathbb{N} \longrightarrow \mathbb{N}$. *Se define la clase* $\boldsymbol{NTIME}(f)$ *como*

$$\boldsymbol{NTIME}(f) = \left\{\mathbb{L} \subseteq \Sigma^* : \exists T_I \text{ con } Res_{T_I} = \chi_{\mathbb{L}} \wedge T_{T_I} \in O(f)\right\},$$

donde T_I *es una máquina de Turing indeterminista.*

Resulta claro a partir de la definición anterior que si $f_1(n) \leq f_2(n)$ para todo $n \in \mathbb{N}$, entonces $\mathbf{DTIME}(f_1) \subseteq \mathbf{DTIME}(f_2)$. Similarmente ocurre con las clases **NTIME**, esto es, si $f_1(n) \leq f_2(n)$ para todo $n \in \mathbb{N}$, entonces $\mathbf{NTIME}(f_1) \subseteq \mathbf{NTIME}(f_2)$. Asimismo, debemos tener presente que las máquinas de Turing deterministas son un caso particular de las indeterministas y, por tanto, dada una función temporalmente computable cualquiera, f, se cumple que $\mathbf{DTIME}(f) \subseteq \mathbf{NTIME}(f)$.

Las clases **DTIME** y **NTIME** nos van a permitir definir otras clases que clasifiquen los problemas de acuerdo a su complejidad de una forma que respete nuestra intuición.

Definición 21. *(Clase **P**)*
Se define la clase de complejidad **P** *como*

$$\boldsymbol{P} = \bigcup_{j=1}^{\infty} \boldsymbol{DTIME}(n^j).$$

Definición 22. *(Clase **NP**)*
*Se define la clase de complejidad **NP** como*

$$\boldsymbol{NP} = \bigcup_{j=1}^{\infty} \boldsymbol{NTIME}(n^j).$$

Volviendo sobre el problema de multiplicar dos números enteros de n cifras, cabe señalar que si se utiliza el método clásico, el que todos conocemos, se requieren $2n^2$ operaciones. En otras palabras, una máquina de Turing con alfabeto $\{0,1,\ldots,9\}$ que reciba una palabra de entrada de longitud $2n$ ha de realizar $2n^2$ operaciones básicas para producir una salida que sea, efectivamente, el producto de dos números enteros. Por lo tanto, el algoritmo tradicional para realizar multiplicaciones pertenece a la clase $\mathbf{DTIME}(n^2)$ y, por tanto, a la clase **P**.

Por su parte, un ejemplo de problema de **NP** es de la *factorización de enteros*, esto es, dados $a, b, m \in \mathbb{Z}$, decidir si existe un número primo $p \in [a,b]$ tal que $p|m$. Este ejemplo resulta muy útil para ilustrar lo que ocurre en realidad con los problemas de **NP**. En la Sección 1.5 presentamos el modelo de la máquina de Turing indeterminista y adelantamos que suponía un aumento de la capacidad computacional. Debemos reparar en que solo las máquinas de Turing deterministas representan algoritmos tradicionales pues en las indeterministas las instrucciones que se han de seguir no quedan determinadas con unicidad a partir de la palabra de entrada. Si uno se ha enfrentado a un problema cualquiera, conoce que es holgadamente más sencillo el verificar si un objeto constituye una solución al mismo que el tratar de obtenerla partiendo de cero. Esto se debe a que el proceso de *verificación* es un mero proceso mecánico ajeno a cualquier atisbo de talento. Con esto en mente, podremos admitir que el indeterminismo aspira a ser una especie de formalización del talento. Exacto, podría decirse que una máquina de Turing indeterminsita es más talentosa que una determinista y, por ello, cabe esperar que resuelva los problemas más eficientemente. Así, si en el problema de factorización de enteros, un ser superior con capacidad *adivinatoria* de una cierta naturaleza que no discutiremos aquí nos hace entrega de otro entero, digamos, q, el cual resulta ser tal que $q|p$, y nos insta a verificarlo mediante, por ejemplo, la división euclídea, el problema habrá pasado de estar en **NP** a estar en **P**. De esta forma, puede decirse que la clase **NP** es la clase **P** dotada de un especial talento adivinatorio. Lo que acabamos de plantear es una de vías que habitualmente se utilizan para comprender lo que supone la clase **NP**.

Es posible que el párrafo anterior resulte esperanzador pues ha aparecido una palabra que resulta tan agradable como es "talento", pero debemos ser conscientes de que tan solo estamos tratando con un modelo. Lo que si que es cierto es que los algoritmos, en el sentido estricto, son realmente necesarios para, al menos, verificar si contamos con la solución de un problema. A pesar de lo arduo de formalizar ideas tan intuitivas como con las que estamos lidiando, se ha convenido que los problemas tratables, esto es, los que pueden ser afrontados mediante algoritmos razonables, son exactamente aquellos para los cuales existe una máquina de Turing capaz de resolverlos con un coste temporal

de orden polinómico respecto a la longitud de la entrada, es decir, precisamente los de la clase **P**. Esto es lo que se conoce como la *tesis de Cobham* y supone que la clase **P** sea especialmente importante, quizás más aún que **NP**, al erigirla como el conjunto de los problemas tratables.

La *tesis de Cobham* cuenta con el respaldo de la robustez del modelo Turing, esto es, del hecho de que se respete la clase **P** aun cuando se permitan generalizaciones del modelo de Turing (máquina universal, distintos alfabetos,...). Sin embargo, es cierto que considerar tratable problemas que se encuentren en $\mathbf{DTIME}(n^{1000000000})$ es cuanto menos acrobático debido a que ni en varias vidas el ordenador más potente con el que contamos hoy en día teminaría su computación. Asimismo, otra de las crítica a la tesis concierne precisamente al encumbramiento del determinismo en detrimento de las posibilidades de otros modelos como el cuántico. En efecto, el problema de la factorización de enteros puede ser resuelto mediante el *algoritmo de Shor* si se admite lo que se conoce como máquina de Turing cuántica. Sin embargo, este modelo no parece ser físicamente realizable, al menos, en la actualidad.

Por otra parte, es claro que $\mathbf{P} \subseteq \mathbf{NP}$ pues cada clase **DTIME** está contenida en su correspondiente clase **NTIME**. De esta manera y tras las consideraciones previas, la intuición nos dice que deberán existir algunos problemas de **NP** que sean verdaderamente difíciles, esto es, que no puedan ser tratables. Esta idea se formaliza mediante la siguiente definición.

Definición 23. *(Problema reducido)*
Sean $\mathbb{L}_1$ *y* $\mathbb{L}_2$ *dos lenguajes asociados a sendos problemas de decisión. Si existe una aplicación* $R : \Sigma^* \longrightarrow \Sigma^*$ *tal que*

$$w \in \mathbb{L}_1 \Longleftrightarrow R(w) \in \mathbb{L}_2,$$

se dice que un problema es reducible *al otro y que es* "al menos tan díficil" *como él.*

Lo habitual es tratar con ciertas reducciones como son la de Levin o la de Karp. Inmiscuirse en los detalles de las mismas queda fuera de los propósitos de este trabajo (ver, por ejemplo, [10]). Sin embargo, si que debemos comentar que, dada una clase de complejidad, se dice que un problema es *duro* cuando todas sus reducciones siguen perteneciendo a la misma clase. De esta forma, se puede hablar de *clases duras* que serán aquellas formadas por los problemas verdaderamente difíciles de una clase dada. Asimismo, se habla de problemas *completos* en una cierta clase cuando todos los problemas de la propia clase son reducibles a él. De esta manera, los problemas completos captan en esencia todo el potencial de la clase y basta con que uno solo de ellos pertenezca a otra clase para que todos ellos lo hagan. En tal caso, se dice que ambas clases *colapsan*. Por su parte, resulta interesante la clase **NP**-dura, pues gran parte de problemas que contiene aparecen en casi cualquier ámbito y, en particular, en el de la lógica. Existen diversas vías para afrontar estos problemas que van desde la *fuerza bruta*, esto es, tratar de

verificar todas las soluciones posibles, hasta tratar de encontrar un algoritmo eficiente, lo que equivaldría a demostrar que el problema pertenece a la clase **P**, pasando por modificar el problema en otro equivalente que resulte ser fácilmente resoluble [3]. Por supuesto, siempre contamos con una última posibilidad: tirar la toalla.

Concluyamos esta sección llamando la atención sobre el hecho de que, tras introducirnos someramente en el ámbito de la teoría de la complejidad de la manera en que lo hacen los matemáticos, hemos acabado por confundir, pese a haber realizado varias llamadas de atención, los conceptos de *problema* y *algoritmo*. Hemos admitido en el párrafo anterior la posibilidad de que un problema verdaderamente difícil pueda ser reducido a uno fácil si se encuentra un algoritmo eficiente que lo resuelva. Entonces, debemos cuestionarnos hasta qué punto el problema realmente era difícil. En efecto, la clasificación de los problemas en base a su complejidad depende de los avances que vayan realizando los matemáticos y no es más que una rémora del anhelo por una concepción platónica de la matemática el tratar de defender una clasificación *a priori* de los problemas conforme a su complejidad.

2.4. ¿P = NP?

Desde que en 1971, tras tratar de demostrar inútilmente que **P** $\subsetneq$ **NP** y postular que **P** = **NP**, la conjetura lanzada por el matemático neoyorquino Stephen Cook no ha hecho más que ganar relevancia. Tanto es así que el Clay Mathematics Institute, ubicado en Cambridge, Massachusetts, ofrece desde el año 2000 un millón de doláres a aquel que sea capaz de dar una respuesta a su conjetura. En efecto, la conjetura de Cook ha sido catalogada como uno de los siete problemas del milenio. Sin embargo, ya anunciamos aquí que, de acuerdo a la opinión de la inmensa mayoría de los expertos en computación, el problema no será resuelto en este siglo.

El esquema de la demostración de que **P** = **NP** habría de ser el siguiente: Encontrar un problema **NP**-duro que esté en **P**, esto es, hallar una máquina de Turing determinista que lo resuelva en tiempo polinómico respecto de la entrada; como el problema es duro, entonces cualquier otro problema de **NP** podría ser reducido a él y entonces tendríamos una máquina de Turing determinista que resuelve cualquier problema de **NP** en tiempo polinómico pero esto es tanto como decir que las clases **P** y **NP** colapsan. En otras palabras, lo que queremos transmitir es que si la conjetura fuese cierta, entonces dispondríamos de un algoritmo capaz de resolver hasta los problemas muy difíciles.

En base a lo anterior, la mayor parte de los artículos escritos por matemáticos

[3] Para que esto sea válido ha de cumplirse que lo que se conoce como *dependencia continua de los datos*, esto es, que si los datos (el problema) varían ligeramente, se espera que la solución varíe de la misma manera. Esta forma de proceder es habitual en el tratamiento de ecuaciones en derivadas parciales. Por ejemplo, no es posible hallar una solución analítica para la ecuación de Black-Schöles con determinadas condiciones pero, en cambio, el problema puede ser reformulado posibilitando que la solución original pueda ser aproximada con el grado de exactitud que se desee.

defienden que no debe creerse que la conjetura es cierta pues supondría, entre otras cosas, que todas las enfermedades conocidas pudiesen ser perfectamente diagnosticadas, que todos los problemas de optimización con los que lidian día a día las empresas pudiesen ser resueltos, así como que la inteligencia artifical podría ser llevada al máximo exponente, permitiendo la creación de máquinas capaces de imitar procesos cognitivos humanos. Seguramente, también pudiesen realizarse predicciones fiables (no probabilísticas) sobre todo cuanto puede ser modelado mediante ecuaciones diferenciales, por ejemplo, el clima, los modelos sociológicos, ciertos aspectos de la economía, y un largo etcétera. Como contrapartida, deberíamos estar dispuestos a renunciar a los sistemas criptográficos que utilizamos en la actualidad, pues están basados en el problema de la factorización de enteros, y por ende, a la intimidad de nuestras comunicaciones.

Por su parte, sería también posible decidir el lenguaje formado por todos los teoremas, de una cierta longitud cualquiera, de una determinada teoría. Esto no es más que una versión finita del *Entscheidungsproblem* y podría tener consecuencias relacionadas con la *omnisciencia lógica*. En efecto, si la conjetura fuese cierta, entonces podríamos conocer automáticamente (en un tiempo polinómico) todas y cada una de las consecuencias lógicas de los axiomas, los cuales podrían ser simples hechos contrastados por la experiencia. Esto resulta inconcebible incluso para el propio Turing quien en [17] defiende que la aceptación de que tan pronto como somos conscientes de un hecho pasamos a ser conscientes de todas sus consecuencias es una asunción muy útil en diversas circunstancias pero falsa. Por su parte, resultaría que la incomputabilidad del *Entscheidungsproblem* no sería tan desalentadora pues los que realmente acaparan nuestro interés son los teoremas que pueden ser demostrados mediante una prueba relativamente tratable en lo que a términos de su longitud se refiere.

Parece ser que la mayor parte de los argumentos mostrados por la comunidad matemática para desestimar la veracidad de la conjetura de Cook están promovidos por el pavor que suscita el pensar que los problemas muy "difíciles" puedan ser resueltos por máquinas. Es más, aun cuando hubiésemos podido probar la falsedad de la conjetura de Cook, seguirían apareciendo interrogantes dignos de tratar. Las máquinas han demostrado en numerosas ocasiones que son más fiables que los humanos en diversos aspectos pero, por ejemplo, no son capaces de conducir vehículos de manera que se garantice la no ocurrencia de accidentes. Recientemente han aparecido en los medios de comunicación noticias tendenciosas calificando de fracaso el que un vehículo no pilotado por un humano haya sufrido un accidente como si los vehículos llevados por personas no los sufriesen. En efecto, cabe preguntarse en primer lugar si los humanos somos capaces de resolver problemas **NP**-duros. De ser así, si la conjetura fuese falsa, podríamos pensar que tenemos un buen argumento para afirmar que los humanos no pueden ser simulados por máquinas de Turing deterministas. En cambio, la experiencia nos dice que, salvo en contadas ocasiones, no existe razón para creer que los humanos pueden resolver tal clase de problemas. Basta atender a que frecuentemente no somos siquiera capaces de resolver correctamente problemas de **P** tales como certificar si un número dado arbitrario es primo.

En cualquier caso, lo que es cierto es que se viene manifestando continuamente un deseo por adaptar la realidad a los modelos matemáticos olvidando que tales modelos

no son mas que eso: modelos. Es inherente a las entretelas de la modelación el asumir determinadas hipótesis a fin de crear un modelo consistente con la siguiente condición: cuantas mas hipótesis se incluyan, menos realistas serán los modelos. Aun cuando esto es aceptado en cualquier rama de la ingeniería y de la física, por alguna razón, parece no aceptarse en el ámbito de la computación. Si reparamos en que todo lo que hemos tratado no es más que un modelo, entonces no cabe alarmarse por el hecho de que la conjetura de Cook pudiese ser verdadera. Efectivamente tendría consecuencias sobre el modelo que es la teoría de la computación y seguramente también ocasionaría efectos sobre determinados aspectos de nuestra vida diaria, pero nunca sobre nuestra condición de seres necesarios para la existencia de los problemas y de las máquinas.

2.5. REFERENCIAS BIBLIOGRÁFICAS.

Introducirse en la teoría de la complejidad es una tarea complicada para cualquiera. Además, este trabajo ha pretendido presentar las nociones fundamentales de manera que resultasen inteligibles incluso a aquellos desprovistos de una vasta formación matemática. Por ello, algunas de las obras que hemos utilizado pueden resultar demasiado densas si no se dispone de un amplio bagaje en el manejo simbólico.

- Los artículos de Pardo ([10]-[11]) realizan una clara exposición de las nociones técnicas de la teoría de la complejidad. En realidad, profundiza muchísimo más de lo que nosotros lo hemos hecho en este trabajo, presentando una ingente cantidad de resultados. Por su parte, quizás adolezcan de una carencia de apuntes filosóficos.
- Por su parte, el artículo de Aaronso ([1]) suple perfectamente la carencia de los artículos a los que nos acabamos de referir. En efecto, Aaronson presenta una amplía variedad de ideas originales al respecto de las consecuencias filosóficas de la complejidad computacional.
- Asimismo, el artículo de Cobham ([2]) también recoge ciertos apuntes que podríamos catalogar como filosóficos al respecto de las distintas clases de complejidad.
- Finalmente, las obra de Davis y Weyuker ([3]) y de Pudlak ([13]) constituyen sendos excelentes manuales que cubren holgadamente todos los aspecto técnicos que hemos tratado.

Comentarios finales.

Como hemos visto a lo largo del trabajo, durante el siglo XX se propusieron diversos modelos computacionales de naturalezas radicalmente diferentes para afrontar el problema de modelizar lo que se entiende por resolver un problema. Así, pronto se planteó la cuestión al respecto de cuál de todos los modelos propuestos era el correcto y si es que alguno habría de merecer tal reconocimiento realmente. Puesto que todos ellos cumplían ciertas propiedades que se habían considerado necesarias (*efectividad* y *completitud*), los interrogantes anteriores fueron remplazados por la cuestión de cuál de todos era el más natural. Por supuesto, la naturalidad resulta ser una propiedad totalmente subjetiva pero, sin embargo, es cierto que propició la aparición de ciertos resultados que, al contrario de lo que suele acontecer en matemáticas, apaciguaron el debate en beneficio del consenso.

En 1934, Kleene intentó demostrar que toda función que se pudiese concebir como computable resultaría ser λ-definible. Tan solo unos meses después, Church conjeturó que las funciones computables eran exactamente aquellas funciones que eran λ-definibles, esto es, que las nociones intuitivas de algoritmo y computación eran capturadas con total precisión por su modelo computacional. A esta suposición se la conoce habitualmente como *tesis de Church*. Aproximadamente dos años después, Turing realizó una conjetura análoga que involucraba al modelo computacional que él mismo había propuesto. Como cabía esperar, a su conjetura se la denominó como *tesis de Turing*.

En 1937, Turing logró demostrar que su modelo computacional y el de Church eran equivalentes en el sentido de que cualquier computación que pudiese ser realizada por uno también podría ser llevada a cabo por el otro. Puesto que el modelo de Turing resultaba ser notablemente menos engorroso, pronto fue aceptado por la comunidad matemática como el idóneo para asir las nociones intuitivas de algoritmo y computación. Esto es lo que se dió en llamar *tesis de Church-Turing*, aunque recientemente el término ha evolucionado en *tesis de la computación* ([14]). Poco a poco se fue demostrando que, en realidad, todos los modelos antes mencionados resultan ser equivalentes en el mismo sentido.

Todo lo anterior nos debe hacer caer en la cuenta de que, independientemente del entorno en el que se conciban los problemas, sus características intrínsecas más esenciales resultan ser similares. En mi opinión, esto se debe a que los problemas, incluso cuando puedan presentarse de las más dispares maneras, dada nuestra condición de seres cognoscitivos necesarios para la existencia de los mismos, deben conservar alguna semejanza subyacente a la hora de ser comprendidos pues, al fin y al cabo, quienes lo hacemos somos todos igualmente humanos. De esta manera, podría utilizarse la tesis de

Church-Turing para aducir que los problemas carecen de una naturaleza objetiva propia pues, de ser así, parece improbable que encajen de la manera en que lo hacen en todos y cada uno de los modelos sugeridos por los distintos matemáticos.

Es un hecho que toda la teoría de la computación surge a partir del inmenso desapego de los matemáticos por las paradojas. Actualmente, algunos investigadores entre los que cabe destacar a Graham Priest, apuntan que posiblemente se deban aceptar las paradojas como objetos inherentes a la propia existencia humana y a sus consiguientes límites. Es un hecho que convivimos con las limitaciones, basta observar que solo disponemos de un tiempo limitado de vida. En esta línea, Priest manifiesta que las paradojas aparecen cuando sobrepasamos los límites del pensamiento y, puesto que las distintas clases de complejidad que hemos tratado parecen pretender lidiar con ciertos límites en cuanto a aquello que podemos resolver, podríamos pensar que, al cuestionarnos si **P** = **NP**, hemos caído en una paradoja justo al trascender precisamente el límite que atañe a la complejidad. Asimismo, en mi opinión la comunidad matemática ha pecado de pesimista, agorera e insidiosa al defender que si la conjetura de Cook fuese cierta, entonces nos veríamos sobrepasados por la capacidad de las máquinas. Debemos recordar el carácter contingente de las mismas y, en consecuencia, reconocer que la capacidad computacional de las máquinas es precisamente la nuestra, pues no constituirían sino una herramienta desarrollada por humanos para obtener soluciones complejas.

Por otra parte, se debe tener presente que hemos ido renunciando a la generalidad de los problemas. Hemos seguido la reducción de problema, problema matemático, problema de decisión previa codificación pertinente, y sin embargo, hemos pretendido extrapolar las consecuencias del modelo como si tal reducción no se hubiese realizado. Quizás se deba a la especial idiosincracia de las matemáticas pero deberíamos, al menos, reflexionar sobre si hemos olvidado la esencia original de los problemas. El nacimiento de la teoría que hemos expuesto estaba ligado a la fundamentación de ciertos conceptos y, como consecuencia del formalismo, esos conceptos abstractos se han distorsionado en un compendio de fórmulas extrañas, perdiendo de vista que tales fórmulas no aspiran a ser más que eso, símbolos formales.

No queremos decir que la teoría de la computación sea inútil. Ni mucho menos. Gracias al desapego de los matemáticos por las paradojas poseemos hoy en día avances tecnológicos que eran impensables años atrás. Pero si que sería adecuado que la comunidad matemática fuese consciente del paradigma filosófico del cual surgió gran parte de los conceptos que se estudian hoy en día. Es posible que la especialización y la prisa a la que nos somete la sociedad constituyan graves impedimentos de cara a realizar tales divagaciones filosóficas, pero debemos promoverlo pues si es cierto que el *programa de Hilbert* no pudo sobreponerse a la fuerza de las contradicciones, no es menos cierto que supuso una suerte de Ítaca para la comunidad matemática y que, aún cuando nunca atracasen en sus aguas, fue el propio viaje lo que realmente resultó enriquecedor *per se*. Por ello, no resulta descabellado plantearse si sería propicio plantear una suerte de *programa de Hilbert* que abrazase las paradojas y que las integrase como parte del propio pensamiento humano. Quien sabe lo que podría deparar ese otro viaje.

A

APÉNDICE.

A.1. SUMAR EN DISTINTOS MODELOS COMPUTACIONALES.

El presente apéndice pretende constituir una suerte de argumento visual que ponga de manifiesto la simplicidad del modelo de Turing frente al resto. Lo que en ningún caso es nuestro objetivo es el de presentar una guía introductoria a los otros modelos computacionales. Para ello, existen obras en la literatura al respecto tales como [6].

MODELO DE KLEENE.

Vamos a tratar de construir la función suma usual

$$S(n,m) := n+m.$$

Para computar $S(n,m)$ deberemos haber computado anteriormente $S(n,m-1)$. Iterando esta idea, es claro que lo primero que se ha de computar es $S(n,0)$. De esta manera, para conseguir construir construir S vamos a requerir de ciertas funciones intermedias, concretamente, $\pi^1_1(x)$, $\pi^3_3(x_1,x_2,x_3)$, $\sigma(x)$, $g_1(x_1,x_2,x_3)$ y $f_4(x_1,x_2)$. El objetivo es llegar a que $S \equiv f_5$. Para ello hay que seguir las reglas de formación relativas a este modelo.

- $f_1 \equiv \pi^1_1(n)$;
- $f_2 \equiv \pi^3_3(n,m,p)$;
- $f_3 \equiv \sigma(n)$;

- $f_4(n,m,p)$ es la composición de f_3 y f_4;
- $f_5(n,m)$ es la composición de f_1 y f_4.

Notemos que

$$f_5(n,m) = f_4(n,m-1,f_5(n,m-1)) = f_5(n,m-1)+1 = \cdots = f_5(n,0)+m = \pi_1^1(n)+m = n+m.$$

Por lo tanto, es cierto que $f_5 \equiv S$ y así la función suma resulta ser computable para el modelo de Kleene.

MODELO DE CHURCH.

Veamos que la función suma usual puede definirse en el λ-cálculo mediante un λ-término. Antes, vamos a presentar las dos reglas de transformación aceptadas en el λ-cálculo:

1. *α-conversion.* Se denota por $\longrightarrow_\alpha$ y sirve para renombrar una determinada variable.
2. *β-contraccion.* Se denota por $\longrightarrow_\beta$ y permite transformar $(\lambda x.M)N$ en un λ-término haciendo $(\lambda x.M)N \equiv M[x := N]$.

Lo que se pretendemos es definir la función suma usual $S(n,m) := n+m$ como un λ-término $S \equiv \lambda abfx.af(bfx)$. Para ello, debemos verificar que, para cuales quiera que sean n y m, se tiene $Sc_nc_m \longrightarrow_\beta \cdots \longrightarrow_\beta c_{n+m}$. Veámoslo.

$Sc_nc_m \equiv (Sc_n)c_m \equiv ((\lambda abfx.af(bfx))c_n)c_m \longrightarrow_\beta (\lambda bfx.c_nf(bfx))c_m \longrightarrow_\beta \lambda fx.c_nf(c_mfx) \equiv \lambda fx.(\lambda fx.f^n x)(f((\lambda fx.f^m x)fx) \longrightarrow_\beta \lambda fx.(\lambda x.f^n x)((\lambda fx.f^m x)fx) \longrightarrow_\beta \lambda fx.(\lambda x.f^n x)((\lambda x.f^m x)x) \longrightarrow_\beta \lambda fx.(\lambda xf^n x)(f^m x) \longrightarrow_\beta \lambda fx.f^n f^m x \equiv \lambda fx.f^{n+m}x \equiv c_{n+m}$, como queríamos probar.

MODELO DE TURING.

Vamos a construir ahora una máquina de Turing que transforme la palabra de entrada $1^{(n)}01^{(m)}$ en $1^{(n+m)}$. Utilizando esta codificación, la máquina sera capaz de computar la función suma usual.

Formalmente, la máquina ha de ser $T = (\{q_1,q_2,q_3\},\{0,1\},\{0,1,\sqcup\},\delta,q_1,\sqcup,\{q_3\})$, donde la función de transición viene dada por

- $\delta(q_1,1) = (q_2,\sqcup,Derecha)$;
- $\delta(q_2,1) = (q_2,1,Derecha)$;
- $\delta(q_2,0) = (q_3,1,Permanecer)$;

- $\delta(q_1, 0) = (q_3, \sqcup, Permanecer)$.

Supongamos que $1^{(n)}01^{(m)}$ es la palabra de entrada y que n es no nulo. La secuencia de configuraciones internas de la máquina sería la siguiente $q_1 1^{(n)}01^{(m)} \longrightarrow q_2 1^{(n-1)}01^{(m)} \longrightarrow 1q_2 1^{(n-2)}01^{(m)} \longrightarrow \cdots \longrightarrow 1^{(n-1)}q_2 01^{(m)} \longrightarrow 1^{(n-1)}q_3 1^{(m+1)}$. Como el estado q_3 es final, la maquina se detiene arrojando la salida $1^{(n-2)}1^{(m)} = 1^{(n+m)}$. Si $n = 0$ y $m \neq 0$, al introducir la palabra de entrada $1^{(n)}01^{(m)}$, se ejecuta una única instrucción con resultado $1^{(m)}$. En cualquier caso, la máquina produce el resultado deseado y, por tanto, la función suma usual resulta ser Turing-computable.

BIBLIOGRAFÍA.

[1] Aaronson, S. (2011). "Why philosophers should care about computational complexity". http://www.scottaaronson.com/blog/?p=735.

[2] Cobham, A. (1964). "The intrinsic computational difficulty of functions". *Proceedings of the 1964 international congress for logic, methodology and philosophy of science*, pp. 24-30.

[3] Davis, M.D. & Weyuker, E.J. (1983). *Computability, complexity and languages*. San Diego, Academic Press.

[4] Gray, J.J. (2006). *El reto de Hilbert*. Crítica.

[5] Hintikka, J. (1962). *Knowledge and belief*. Cornell University Press.

[6] Kleene, S.C. (1936). λ-definability and recuriveness. *Duke Mathematical Journal*.

[7] Kolmogórov, A.N., Dragalin, A. G. (2013). *Lógica matemática. Introducción a la lógica matemática*.Traducción del ruso por Carlos Daniel Navarro Hernández y Juan Enrique Palomino Pérez. Hayka libros.

[8] Kolmogórov, A.N., Dragalin, A. G. (2013). *Lógica matemática. Capítulos complementarios*.Traducción del ruso por Carlos Daniel Navarro Hernández y Juan Enrique Palomino Pérez. Hayka libros.

[9] Martin, J. C. (2003). *Lenguajes formales y teoría de la computación*. McGraw-Hill.

[10] Pardo, L.M. (2012). "La conjetura de Cook (¿P=NP?). Parte I: Lo básico". *La gaceta de la RSME*, Vol. 15, No. 1, pp. 117-147.

[11] Pardo, L.M. (2012). "La conjetura de Cook (¿P=NP?). Parte II: Probabilidad, interactividad y comprobación probabilística de las demostraciones". *La gaceta de la RSME*, Vol. 15, No. 2, pp. 303-333.

[12] Priest, G. (2002). *Beyond the limits of thought*. Oxford University Press.

[13] Pudlak, P. (2013). *Logical foundations of mathematical and computational complexity*. Springer.

[14] Robic, B. (2015). *The foundations of computability theory*. Springer.

[15] Salomaa, A. (1973). *Formal languages*. Nueva York, Academic Press.

[16] Shor, P.W. (1997). "Polynomial-time algorithms for prime factorization and discrete logarithms on a quamtum computer". *SIAM J. Comput*, Vol. 26, No. 5, pp. 1484-1509.

[17] Turing, A.M. (1936). "On computable numbers, with an application to the Entscheidungsproblem". *Proceedings, London Mathematical Society*, Vol. 2, No. 42, pp. 230-265.

Lógica y Filosofía de la Ciencia

Máster Interuniversitario

Declaración de integridad intelectual

Trabajo Fin Máster
Curso 2016-2017

Título del trabajo: Panorámica de la teoría computacional: Desarrollo y problemas abiertos.

1. Sé que copiar es una forma de deshonestidad académica.
2. He leído el documento sobre cómo ser intelectualmente íntegro, estoy familiarizado con sus contenidos y he evitado todas las formas de plagio allí recogidas.
3. Cuando utilizo las palabras de otros, lo indico mediante el uso de comillas.
4. He referenciado todas las citas e igualmente el resto de ideas tomadas de otros.
5. No he plagiado mi propio trabajo.
6. No permitiré a otros que plagien mi trabajo.

Fecha: 10 de julio de 2017.

Firma: