JORNADAS SOBRE LOS
PROBLEMAS DEL MILENIO
Barcelona, del 1 al 3 de junio de 2011

P VERSUS NP

LUIS M. PARDO

Resumen. Estas páginas sólo pretenden ser un resumen minimalista de algunos aspectos de la conjetura de Cook–Levin–Karp. No son completas ni equilibradas y están sesgadas por una elección personal, muy subjetiva, para orientarse hacia la reciente prueba del Teorema **PCP** por I. Dinur. Las limitaciones del modelo de curso propuesto, hacen también que estas notas pretendan un salto de máxima dificultad por pasar del magro conocimiento de un alumno de "segundo ciclo" de Matemáticas a un tema de investigación, en tres sesiones de cincuenta minutos. El texto comienza desde las nociones más elementales (máquina de Turing), pasando por las definiciones y propiedades básicas de las clases de complejidad computacional, hasta poder desencriptar el significado formal de la pregunta "¿**P** = **NP**?" y concluir con algunos aspectos de interactividad y el Teorema **PCP**. Si bien buena parte de los contenidos (el primer tercio, esencialmente) son, usualmente, de conocimiento común a toda formación como Ingeniero Informático, no es el caso de la formación de los Matemáticos, lo que justifica esta propuesta. La atracción por todo problema matemático es, obviamente, subjetiva y dependiente de modas y épocas. Pero hay algo que hace de esta pregunta (¿**P**= **NP**?) un sujeto especial en la historia de la aplicabilidad de la Matemática: Es una pregunta matemática procedente de otro ámbito: la Informática. Sin embargo, como trataremos de exponer a lo largo del texto, *y por vez primera en la historia de la modelización, no es la matemática la que intenta modelizar la realidad, sino que es la realidad la que trata de imitar a la abstracción matemática.* El papel del conocimiento matemático se ve, por tanto, reforzado por este tipo de problemas. El material expuesto no es original. Es una mera exposición de resultados conocidos por los especialistas del ámbito científico. Lo único que se puede achacar al autor es la elección del orden y los temas destacados. Otras opciones hubieran sido igualmente razonables.

"...En un mot, les calculs sont impraticables!!"[1]
É. Galois, 1832

1. Introducción

Permítase al autor unas pocas frases iniciales que avisen al lector del contexto en el que se elaboran estas notas. Estas páginas son solamente unas notas para un curso breve alrededor de un problema abierto (¿**P**=**NP**?), con indicaciones de alguno de los resultados recientes que más han impactado a la comunidad científica

Financiado por el proyecto MTM2010-16051.

[1] "...En una palabra, ¡los cálculos son impracticables!"

y sólo pretenden ser un resumen minimalista de algunos aspectos de la conjetura de Cook–Levin–Karp. No son completas (sería imposible citar toda la bibliografía sobre el tema), no son equilibradas (hay demasiados aspectos que deben quedar fuera) y están sesgadas por una elección personal de quien suscribe (de orientarse hacia la reciente prueba del Teorema **PCP** por I. Dinur). Las limitaciones del modelo de curso propuesto hacen que sea necesario comenzar con las definiciones más elementales (que no forman parte del magro –cuando no nulo– conocimiento en Modelos de Computación que contiene la formación habitual de un alumno de Matemáticas) para realizar un salto de máxima dificultad y terminar en un tema reciente de investigación. Si bien las notas que siguen alcanzan algunos aspectos de la frontera del conocimiento en Informática Teórica, no alcanzan, en ningún caso, límites que no sean del conocimiento común del iniciado.[2] A veces se llega a detallar una prueba, otras apenas se indica y, en la mayoría de los casos, habrá que orientar al lector hacia referencias bibliográficas no siempre accesibles.

La Complejidad Computacional es una noción nueva, reciente para la Matemática. Parte del principio obvio de que no basta con disponer de un algoritmo que resuelva un problema. Se necesita, además, que el comportamiento del algoritmo (en términos de tiempo de ejecución y espacio/memoria requeridos) sea "razonable" en relación con el tiempo de vida humano y la velocidad de las máquinas disponibles. Es relativamente sencillo diseñar problemas matemáticos cuyo tiempo de ejecución supere todas las expectativas de vida del Universo (en su modelo actual) o, incluso, del número total de partículas existentes en él. Ejemplos sencillos en eliminación de cuantificadores (tanto en el caso de cuerpos reales como de cuerpos algebraicamente cerrados) o problemas de pertenencia a ideales del anillo de polinomios, rápidamente requieren tiempo doblemente exponencial y/o espacio expo-polinomial (véanse [Ma-Me, 82], [Da-He, 88], o los *surveys* [Prd, 95], [Be-Prd, 06], por ejemplo, sobre la complejidad de los problemas de Eliminación Geométrica). Interpretemos estos resultados a la luz del ordenador más rápido existente. Según el *Top500*, en 2011 estamos hablando del ordenador chino **Tianhe-1A** cuya velocidad máxima se puede estimar en unas $2^{32} \equiv 4 * 10^{15}$ operaciones bit por segundo. Esto significa que para $n = 10$ variables (lo que representa menos de un "toy example"), el tiempo de ejecución de cualquier algoritmo que resuelva cualquiera de esos problemas en el **Tianhe-1A** es de 2^{940} años, lo que en tiempo más comprensible supera los 2^{920} millones de años... Si el lector considera que estos ejemplos clásicos de Teoría de la Eliminación Geométrica son excesivos por sofisticados, pongamos un ejemplo mucho más simple como la ejecución del siguiente algoritmo en el **Tianhe-1A**:

$z := 2$
$i := 1$
while $i \leq 159$ **do**

---

[2]El iniciado en Complejidad Computacional, no encontrará en estas notas del curso nada que no le sea conocido. No se trata de un curso para especialistas, sino para una audiencia desconocedora de los rudimentos mínimos de la Complejidad Computacional.

$z := z^2$, $i := i + 1$
od
OUPUT: z

En algún momento de su ejecución, el ordenador deberá escribir los dígitos de $2^{2^{158}}$, lo que le exigirá realizar aproximadamente 2^{158} operaciones bit, lo que le supondrá, obvia división (velocidad:=espacio/tiempo) un tiempo de unos 2^{100} **años**, lo cual no es muy esperanzador para quien quiera ver el resultado de estos cálculos en ese ordenador.

De entre todos los problemas posibles de Complejidad Computacional y Fundamentos de Matemáticas Computacionales, el abanderado es la conjetura de Cook, que se puede "encriptar" fácilmente (según el código introducido en [Karp, 72]) como:

Problema Abierto 1.1 (CONJETURA DE COOK)**.** *Decidir si el contenido siguiente es estricto:*

$$\mathbf{P} \subseteq \mathbf{NP}.$$

El problema es, en otras ocasiones, encriptado como $\mathbf{P} = \mathbf{NP}$? En estas notas trataremos de desencriptar este código, explicando el significado de la pregunta y mostrando algunos resultados significativos de su contexto. Para empezar debe señalarse que la pregunta es también enunciable como la conjetura de Cook–Levin y, si hemos de ser completos, queda más justa como la conjetura de Cook–Levin–Karp, por ser R. Karp quien la formaliza en su forma actual.

Dejando para más adelante la significación de esta críptica forma, digamos que la relevancia de esta pregunta se basa en tres patas. De una parte, la relativa simplicidad del enunciado, frente a la dureza de su resolución. Hordas de especialistas en Informática Teórica han atacado el problema, algunos con resultados muy significativos que, sin embargo, no han roto la barrera de la respuesta (entre ellos destaca el Teorema **PCP** del que hablaremos al final de estas notas). Se trata de un problema difícil para el que, según parece, la Matemática aún no está preparada. Personalmente, tiendo a creer en la respuesta negativa (como la mayoría) sin tener una razón consciente para defenderla y con la convicción de que no estamos aún preparados para la respuesta. De todos modos, a modo de ejemplo del delirio que suscita la pregunta, permítaseme indicar aquí la política editorial de la principal revista de la principal asociación de Informática del mundo (la ACM) en relación con la conjetura de Cook:

- **Journal of the ACM, P/NP Policy.**[3] *The Journal of the ACM frequently receives submissions purporting to solve a long-standing open problem in complexity theory, such as the P/NP problem. Such submissions tax the voluntary editorial and peer-reviewing resources used by JACM, by*

[3]Tomado de la página oficial de instrucciones para los autores: `http://jacm.acm.org/instructions/pnp`.

requiring the review process to identify the errors in them. JACM remains open to the possibility of eventual resolution of P/NP and related questions, and continues to welcome submissions on the subject. However, to mitigate the burden of repeated resubmissions of incremental corrections of errors identified during editorial review, JACM has adopted the following policy:
No author may submit more than one paper on the P/NP or related long-standing questions in complexity theory in any 24 month period, except by invitation of the Editor-in-Chief. This applies to resubmissions of previously rejected manuscripts.

Mi modesta contribución como editor del *Journal of Complexity*, también me permite conocer la experiencia de equivocadas[4] contribuciones que pretenden haber resuelto esta conjetura con la pretensión de ser publicada en nuestra revista. Los anuncios de resolución deben, por tanto, ser tomados con pinzas[5] quirúrgicas.

El segundo punto de apoyo para el interés de la conjetura de Cook es una suerte de ubicuidad que ya estaba presente desde el mismo momento de su irrupción. A modo de ejemplo, el clásico [Ga-Jo, 79] ya incluye más de 350 ejemplos de problemas **NP**–completos provenientes de ámbitos que van desde la Lógica, a la Teoría de Grafos, la Optimización o la Teoría de Números. Fácilmente se añaden más tarde ejemplos provenientes del Álgebra Conmutativa, la Geometría Algebraica, el Análisis Numérico, la Geometría Diofántica, la Criptografía, los Códigos Correctores de Errores o la Teoría de Juegos, por poner sólo unos ejemplos.

El tercer apoyo, y el principal para los Matemáticos, es la aparición de esta conjetura de Cook–Levin en las dos listas principales de problemas abiertos para las Matemáticas del siglo XXI. Desde la famosa conferencia de D. Hilbert en el París del 1900, 23 problemas han ido guiando buena parte de las Matemáticas del siglo XX. Tal ha sido la influencia de esta lista que nos hemos vuelto aficionados a listas orientativas de problemas emitidas por eminentes matemáticos. Al final del pasado siglo, dos listas de problemas se convierten en las herederas naturales de la lista de Hilbert (sin óbice de la existencia de otras): la lista de Instituto Clay y la lista de 18 problemas de S. Smale en [Smale, 00]. La primera está esencialmente subsumida en la segunda y ambas contienen la conjetura de Cook como uno de los problemas más relevantes para las Matemáticas del presente siglo. Si acaso, debe destacarse un mayor énfasis en la Complejidad Computacional y la Algorítmica como "leitmotiv" de la lista de Smale. Podemos señalar que ambas listas ya tienen problemas resueltos: así, destacar la resolución de la conjetura de Poincaré por G. Perelman, presente en ambas listas, y la resolución de los Problemas **12** (en [Bo-Cr-Wi, 09]), **14** (en [Tu, 02]) y **17** ([Be-Prd, 09a],[Be-Prd, 11] y el *survey* [Be-Prd, 09b]). En ambas sigue, imperturbable, la conjetura de Cook–Levin como problema abierto.

[4]Usando argumentos desorientados basados, por ejemplo, en la resolución de un cubo de Rubik $3 \times 3 \times 3$.

[5]Incluido el caso mucho más serio de Deodalikar el pasado año 2010

Es fácil trazar hacia el pasado los antecedentes de la conjetura de Cook–Levin en la historia de las Matemáticas. El referente histórico natural es el Teorema de los Ceros de Hilbert–Kronecker (**HN**:= Hilbert Nullstellensatz). Por ejemplo, en su forma de Identidad de Bézout, toma la forma siguiente:

Teorema 1.2 (Hilbert Nullstellensatz)**.** *Sea K un cuerpo y $\mathbb{K}$ su clausura algebraica. Sean $f_1, \ldots, f_m \in K[X_1, \ldots, X_n]$ polinomios con coeficientes en K y sea $\mathfrak{a}$ el ideal que generan en el anillo $K[X_1, \ldots, X_n]$. Definamos $V(\mathfrak{a}) \subseteq \mathbb{K}^n$ la variedad de sus ceros, es decir,*

$$V(\mathfrak{a}) := \{x \in \mathbb{K}^n \ : \ f(x) = 0, \ \forall f \in \mathfrak{a}\}.$$

Entonces son equivalentes $V(\mathfrak{a}) = \emptyset$ y $1 \in \mathfrak{a}$. O, equivalentemente, $V(\mathfrak{a}) = \emptyset$ si y solamente si existen $g_1, \ldots, g_m \in \mathbb{K}[X_1, \ldots, X_n]$, tales que

$$1 = g_1 f_1 + \cdots + g_m f_m. \tag{1}$$

Este enunciado es doblemente debido a D. Hilbert[6] y L. Kronecker.[7] Lo que se pretende esencialmente es controlar si las ecuaciones $f_1 = 0, \ldots, f_m = 0$ son satisfactibles sobre la clausura algebraica $\mathbb{K}$ del cuerpo de coeficientes. Es decir, se trata del diseño de algoritmos que resuelvan la siguiente pregunta:

$$\exists x_1 \in \mathbb{K}, \ldots, \exists x_n \in \mathbb{K}, \ f_i(x_1, \ldots, x_n) = 0.$$

Si alguien quisiera argumentar que, al contrario que Kronecker, Hilbert no está interesado en la algorítmica subyacente, bastaría con remitirle a la alumna de D. Hilbert G. Hermann, quien dedica su tesis al Nullstellensatz Efectivo[8] que conducirá a los algoritmos para el tratamiento simbólico del Nullstellensatz y el Problema de Consistencia (dentro de los Métodos Efectivos en Geometría Algebraica), y a posteriores estudios con estimaciones más finas tanto en el caso Efectivo (estimaciones de grado) como en el Aritmético (estimaciones de tallas de coeficientes en el caso diofántico y racional).

De hecho, el principal enunciado en [Cook, 71] (*"*`3SAT` *es* **NP***–completo"*, ya traducido al lenguaje moderno, establecido en [Karp, 72]), consiste en demostrar que el Nullstellensatz de Hilbert es **NP**–duro y que es **NP**–completo si supongo, por ejemplo, que los grados están acotados por 3, el cuerpo es un cuerpo finito $K = \mathbb{F}_2 = \{0, 1\}$ y que la lista de ecuaciones dadas incluye como sublista la lista:

$$X_1^2 - X_1 = 0, \ldots, X_n^2 - X_n = 0.$$

Es decir, el principal enunciado de la conjetura de Cook (del que irán derivando todos los demás) no es sino un problema relativo a la complejidad computacional de una subclase de instancias del Nullstellensatz de Hilbert, con grado acotado, sobre un cuerpo finito. A sabiendas de que ésta no es la presentación más habitual

[6] D. Hilbert. "Über theorie der Algebraischen Formen". *Math. Ann.* **36** (1890) 473–534.

[7] L. Kronecker. "Grundzüge einer arithmetischen theorie de algebraischen grössen". *J. reine angew. Math.*, **92** (1882) 1–122.

[8] G. Hermann. "Die Frage der endlich vielen Schritte in der Theorie der Polynomideale", *Math. Ann.* **95** (1926) 736–788.

de la conjetura de Cook, es apropiado señalarlo tanto porque *"...Caesaris Caesari"* como por ser el antecedente natural no siempre reconocido.

En la perspectiva algorítmica de Kronecker y en la más axiomática de Hilbert, faltan aún ingredientes fundamentales para contemplar esta conjetura de Cook: la *Complejidad Computacional*. El lector podría pensar que el término "Computacional" hace imposible la existencia de antepasados de la noción, previos a la era digital/computacional en la que vivimos. Nada más lejos de la verdad. De nuevo, si hemos de cumplir con la historia habrá que darle a E. Galois la paternidad del concepto, dado que es él quien primero escribe, negro sobre blanco, la cuestión de la tratabilidad algorítimica de un problema. En sus memorias, transcripción de su testamento la víspera del famoso duelo, podemos leer:

> *...Si maintenant vous me donnez une équation que vous aurez choisie à votre gré et que vous désiriez connaître si elle est ou non soluble par radicaux, je n'aurai rien à y faire que de vous indiquer le moyen de répondre à votre question,... sans vouloir charger ni moi ni personne de le faire. En un mot, les calculs sont impraticables!!*[9]

En realidad Galois se refiere a un problema cuyos algoritmos tratables tardarán aún 150 años en aparecer: la factorización de polinomios univariados por algoritmos tratables que demostrarán A.K. Lenstra, H.W. Lenstra y L. Lovasz en 1981 [10] y su generalización a extensiones finitas de cuerpos primos (y, por tanto, separables). Con todo, ya en el siglo XIX aparece el primer resultado sobre la complejidad de un problema: G. Lamé[11] demuestra en 1844 que el número de divisiones realizadas por el algoritmo de Euclides para hallar el máximo común divisor de dos número enteros está acotado por el máximo de sus tallas (en codificación binaria o decimal) que es el máximo de sus logaritmos. En otras palabras, no debe usarse factorización sino Euclides para el cálculo del máximo común divisor de dos números enteros.

Pero para tener una buena modelización de la conjetura de Cook habremos de tener un mayor detalle y precisión en las nociones involucradas. Así, dedicaremos algunas páginas de la primera parte de estas notas a fijar con detalle lo que no es habitual en la formación del Matemático: las nociones de máquina de Turing, las funciones de complejidad en tiempo y en espacio y las clases que determinan.

No es la conjetura de Cook el único o el más relevante de los problemas abiertos en complejidad computacional (aunque sí sea el más afamado). Puestos a elegir me

---

[9]Si usted me diera una ecuación que hubiera escogido a su gusto y de la cual usted desearía conocer si es o no resoluble por radicales, yo no tendría nada que hacer salvo indicaros el modo de responder a esta cuestón ... sin desear encargarme yo mismo, ni encargar a nadie, el hacerlo [responder, con cálculos, a la pregunta]. En una palabra, ¡los cálculos son impracticables!

[10]A. K. Lenstra, H. W. Lenstra, Jr., L. Lovász."Factoring polynomials with rational coefficients".*Mathematische Ann.* **261** (1982) 513–534.

[11]G. Lamé. "Note sur la limite du nombre des divisions". *C. R. Acad. Sci. Paris* Ser. A-B **19** (1844), pp. 867–869.

sentiría más a gusto con una exposición sobre el "smoothed analysis" de D. Spielman (premio Gödel 2008), el test de primalidad de Agrawal, Kayal y Saxena (premio Gödel 2006) o cualquier otro problema en las lista de problemas para la Matemática del siglo XXI que propone S. Smale en [Smale, 00] o, por ejemplo, el premio Gödel 2010 para Arora y Mitchell por sus trabajos de 1998–1999, mostrando un esquema aproximativo en tiempo polinomial (PTAS) para el Problema del Viajante Euclídeo (que no es **NP**–duro). Pero el objetivo del curso es la conjetura de Cook, por estar explícitamente enunciado en la lista del Instituto Clay. Desde mi punto de vista, la jerarquía propiciada por los Teoremas **PCP** es la secuencia alternativa más interesante y sorprendente de las caracterizaciones de la clase **NP**. Aunque el Teorema original supuso ya el premio Gödel para Arora, Sudan y otros en 2001, la "novedad" es la reciente prueba del Teorema **PCP** propuesta por Irit Dinur en su trabajo [Dinur, 07]. Un interesante texto para complementar el curso es el trabajo [Ra-Su, 07].

En todo caso, estas son notas de un curso, nunca un trabajo de investigación original ni un *survey* mesurado, ni completo, sobre las principales líneas de investigación en Complejidad Computacional. Ni puede serlo con las condiciones de contorno propuestas. El siguiente Índice puede ser de utilidad para que el lector siga la deriva de las páginas que siguen:

Índice

2. Un poco de historia

Los términos "algoritmo", "guarismo" y "álgebra" tienen un origen común en los trabajos del matemático uzbeko del siglo XI Muhammad ibn–Musa Al–Juaritzmi quien escribió su tratado *"Hisab al–jabr wa–al–muqabala"* (traducido libremente por Libro (o Tratado) sobre las operaciones "jabr" (restablecimiento) y "qabala" (reducción). Y a la exposición de métodos de manipulación de números y ecuaciones estaba dedicado este tratado. No se trata de una obra original, sino de un compendio del conocimiento combinado de las matemáticas helenísticas y la teoría de números conocida en la India. El libro está fundamentalmente dedicado a la resolución sistemática de ecuaciones de primer y segundo grado, ciencia que se considera independiente. Así son resueltas, por ejemplo, las siguientes "clases" de ecuaciones (como si fueran elementos distinguidos):

$$ax = b, \quad x^2 + bx = a, \quad ax^2 = b,$$

$$x^2 + a = bx, \quad ax^2 = bx, \quad bx + a = x^2.$$

Pensemos que aún no se usan los números enteros, que serán una aportación de las matemáticas del Renacimiento. Otra obra de Juaritzmi (traducido por guarismo esta vez) *"Sobre los números hindúes"*, versión latina del siglo XII, transfiere a las matemáticas europeas la representación de los números enteros en base decimal.

Desde entonces, pasando por la tradición renacentista de resolución de sistemas de ecuaciones polinomiales (Del Ferro, Tartaglia, Viéte), hasta los padres de la Teoría de la Eliminación del siglo XIX (Bézout, Cayley, Hilbert, Kronecker,...) la interacción entre álgebra, algoritmos y ecuaciones ha sido parte integrante de la historia de la Matemática.

Los algebristas no estaban sólo interesados en disquisiciones teóricas sobre propiedades de estructuras, comúnmente llamadas algebraicas, sino también en la resolución de problemas bien concretos: ecuaciones polinomiales univariadas.

Además, desde el Renacimiento, los matemáticos tratan de construir máquinas que les resuelvan las tareas (véase la máquina aritmética de Pascal, por ejemplo).

Sin embargo, debemos señalar que *nadie sabía qué era un algoritmo*. Definiciones del tipo *algoritmo es una fórmula* o *una serie de cálculos finitarios*, o extravagancias imprecisas de parecida superficialidad, eran moneda de cambio entre matemáticos reputados, muy delicados en el manejo de definiciones altamente sofisticadas.

D. Hilbert propone el décimo de sus famosos 23 problemas en la conferencia inaugural del Congreso Internacional de Matemáticos de París del año 1900.[12] A la vista de su conocimiento del Nullstellesatz, D. Hilbert no esconde ninguna intención próxima a lo que sucedió. El famoso Décimo Problema de Hilbert se enuncia del modo siguiente:

Problema 2.1 (Problema X de Hilbert)**.** *Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers.*

En otras palabras,

Problema 2.2 (Problema X de Hilbert)**.** *Dar un algoritmo que realice la siguiente tarea:*
Dada una lista de polinomios con coeficientes enteros $f_1, \ldots, f_s \in \mathbb{Z}[X_1, \ldots, X_n]$*, decidir si*

$$\exists x := (x_1, \ldots, x_n) \in \mathbb{Q}^n, \;\; f_1(x) = 0, \ldots, f_s(x) = 0.$$

Nótese que reducir a una sola ecuación es obvio mediante la suma de los cuadrados $f := f_1^2 + \cdots + f_s^2$. El problema enlazaba con el Nullstellensatz de Hilbert–Kronecker:

Problema 2.3 (Decisión en el Nullstellensatz)**.** *Dados polinomios* $f_1, \ldots, f_s$ *en* $\mathbb{C}[X_1, \ldots, X_n]$*, decidir si:*

$$\exists x := (x_1, \ldots, x_n) \in \mathbb{C}^n, \;\; f_1(x) = 0, \ldots, f_s(x) = 0.$$

Nótese que ahora ya no es posible reducir al caso de una sola ecuación (Macaulay) y que G. Hermann propone un algoritmo basado en las cotas de grado del Nullstellensatz Efectivo. A la sazón, L. Kronecker había introducido años antes un algoritmo para resolver tal problema en su trabajo de 1882.[13]

Si, por el contrario, se preguntara sobre la existencia de raíces reales ($\exists x = (x_1, \ldots, x_n) \in \mathbb{R}^n$) o racionales no se conocía ningún algoritmo en la época en que Hilbert enuncia su famoso problema.

El caso real se resolvió pronto. En 1931, el matemático polaco A. Tarski anuncia que tiene un algoritmo para decidir si una o varias ecuaciones polinomiales poseen solución real (en $\mathbb{R}^n$). Esto aparece publicado en su trabajo.[14] Las circunstancias del ascenso del nazismo en Alemania, la invasión de Polonia y la emigración de Tarski a los Estados Unidos, pospuso la publicación del detalle hasta la aparición

[12] D. Hilbert. "Mathematische Probleme". *Archiv für Mathematik und Physik* **1**(1901) 44–63 y 213–237. Véase también la versión inglesa en D. Hilbert "Mathematical Problems". *Bull. of the A.M.S.* **8** (1902) 437–479.

[13] L. Kronecker. "Grundzüge einer arithmetischen theorie de algebraischen grössen". *J. reine angew. Math.*, **92** (1882) 1–122.

[14] A. Tarski. "Sur les ensembles définissables de nombres réeles". *Fund. Math.* **17** (1931) 210–239.

de una edición preparada por J.C.C. MacKinsey.[15] En el entorno de los años 50, A. Seidenberg publica su propio algoritmo para resolver el caso real.[16]

Resueltos el caso complejo y el caso real, queda el caso diofántico. Para ello, unos pocos matemáticos reflexionan sobre el problema desde un sentido opuesto: si no se conoce la noción de algoritmo poco o nada se puede reflexionar sobre el Problema X. Por tanto, es sobre la noción de algoritmo sobre la que vuelcan sus esfuerzos. En 1916, el matemático noruego A. Thue introduce sus sistemas de reescritura que serán pronto vistos como insuficientes para modelizar el concepto de algoritmo, aunque serán recuperados por Chomsky para su clasificación de los lenguajes formales (gramáticas formales).

Hacia mediados de los años 1930, dos figuras relevantes aparecen para fijar la noción de algoritmo: al austríaco K. Gödel y el británico A. Turing. Rodeados de las figuras de A. Church y su alumno S.C. Kleene.

Hay que hacer referencia al Círculo de Viena, donde destacan la dirección de Hahn o la eventual participación de Tarski, al que Gödel se incorpora. Es en este ambiente donde K. Gödel elabora su famosa tesis (23 páginas) en la que demuestra la Incompletitud de la Teoría Elemental de Números.[17] Aquí Gödel usa por primera vez una noción que tiende a las funciones computables (él las llamó "rekursiv"). Hoy son llamadas "funciones primitivas recursivas", i.e. sin el operador de minimización.

Durante los años 30, K. Gödel visita Princeton varias veces, hasta su traslado definitivo en 1940. En 1934, durante una de sus visitas, dio una charla cuyas notas circularon. Estas notas[18] fueron tomadas por S.C. Kleene y Rosser, quienes a la sazón completaban sus estudios de doctorado bajo la dirección de A. Church. En esta conferencia, Gödel introduce la noción de computabilidad efectiva. Notó que formas más generales de recursión deberían ser admitidas antes de que sus funciones "rekursiv" pudieran cubrir todo lo computable. Así definió una clase que llamó "funciones generales recursivas" (al parecer, sugerido por una carta de Herbrand). Había nacido la noción de algoritmo.

A. Church dirige la tesis de S.C. Kleene sobre una noción de recursividad alternativa a la noción introducida por Gödel. Se trata de la noción de $\lambda-$*calculus*. En los trabajos de Kleene[19] y Church[20] demuestran que su noción de algoritmo y la noción propuesta por K. Gödel son exactamente la misma. Está naciendo la **Tesis de Church**.

---

[15]A. Tarski. *"A decision method for elementary algebra and geometry".* (Prepared for publication by J.C.C. Mac Kinsey, Berkely (1951).

[16]A. Seidenberg. "A new decision method for elementary algebra". *Ann. of Math.* **60** (1954) 365–374.

[17]K. Gödel. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I". *Monatsh. Math. Phys.* **38** (1931) 173–198.

[18]K. Gödel. "On undecidable propositions of formal mathematical systems". In *The undecidable, Basic Papers on Undecidable Propositions, Unsolvable Problemas and Computable Functions*, Raven Press, Hewlett, NY (1965) 41–71.

[19]S.C. Kleene. "$\lambda-$definability and recursiveness". *Duke Math. J.* **2** (1936) 340–353.

[20]A. Church. "An unsolvable problem of elementary number theory". *Am. J. Math.* **58** (1936), 345–363.

La tesis de Church no es propiamente una Tesis, ni un Teorema, sino una Definición:

Se llama algoritmo a toda función recursiva, todo procedimiento del $\lambda-$calculus y toda noción equivalente a ambas.

Por su parte, A. Turing basó gran parte de su investigación en la interacción entre Matemáticas y la naciente Informática. Por ejemplo, en su trabajo de 1948 A. Turing[21] introducirá la *noción de condicionamiento del Álgebra Lineal Numérica*, convirtiéndose en el fundador del Álgebra Lineal Numérica moderna (en los textos al uso se concede la prioridad a J. von Neumann y colaboradores o a autores como Wilkinson, olvidando la contribución esencial de Turing).

A. Turing publicaba su modelo en su trabajo de 1936[22] dedicado a caracterizar los números reales "definibles" (recursivamente enumerables) y, como Gödel, fue invitado a visitar Princeton entre 1936 y 1938, donde defenderá su tesis introduciendo las máquinas de Turing con oráculos. Probó en un apéndice la equivalencia entre su modelo y la $\lambda-$definibilidad. De hecho, dos son las aportaciones fundamentales de Turing en este trabajo del 1936. De una parte, la introducción de un nuevo modelo alternativo de algoritmo (máquinas de Turing) y el resultado de autorreducibilidad basado en la máquina Universal. De otra, el análisis de los números reales recursivamente enumerables que influirá en los fundamentos del Análisis por parte de los "constructivistas". Aquí recogeremos un poco de ambas ideas.

Emil Post también introdujo su modelo de cálculo en 1936, que resultó equivalente al de Turing,[23] cuyo formalismo ha influenciado fuertemente los formalismos introducidos a posteriori. Post llegó a describir *la tesis de Church como una ley natural, "un descubrimiento fundamental" concerniente a "the mathematizing power of Homo Sapiens"*. Así, la Tesis de Church toma la forma siguiente:

Definición 2.4 (TESIS DE CHURCH)**.** *Llamaremos computable a toda función calculable por alguno de los siguientes métodos equivalentes de caracterización:*

- *Calculable por una máquina de Turing,*
- *es una función general recursiva,*
- *es $\lambda-$definible,*
- *es Post–calculable,*

o calculable por cualquier otro procedimiento equivalente a alguno de éstos.

El modelo de Turing es, con mucho, el de mayor sencillez definitoria, pero también el más relacionado con algo que no soñaban en aquélla época: los ordenadores.

---

[21] A. Turing. "Rounding-off errors in matrix processes". *Quart. J. Mech. Appl. Math.* **1** (1948), 287–308.

[22] A. Turing. "On computable numbers, with an application to the Enscheidungsproblem". *Proc. London Math. Soc., Ser. 2,* **42** (1936) 230–265.

[23] E. Post. "Finite Combinatory processes–formulation I". *J. Symbolic Logic* **1** (1936), 103–105.

2.0.1. Un Inciso Histórico: Computación y Criptografía, desde el principio de los tiempos. De hecho, existe una historia, cuyos datos no han sido revelados hasta finales de los años 1990, que explica la decisión de aplicar el modelo de Turing a las arquitecturas de los ordenadores más antiguos. Permítame el lector esta digresión que, por otro lado, es accesible en cualquier referencia habitual.

Turing es un joven matemático brillante a finales de los años 1930 cuando comienza la Segunda Guerra Mundial. En esa época, algunos matemáticos polacos han logrado descubrir el sistema de comunicación secreto del ejército alemán: la "famosa" máquina Enigma con tres rotores. A partir de 1939, de poco le sirve a Polonia esta información y es Churchill quien, personalmente, se hace cargo del rescate de algunos de estos matemáticos y los traslada a Inglaterra. Allí, manteniendo el máximo secreto, Churchill ha creado el proyecto "Ultra" en un lugar aislado de la campiña británica (conocido como Bletchley Park). Entre los miembros del proyecto "Ultra" se encuentra A. Turing quien acabará dirigiendo el equipo de descodificadores. Tras conseguir la información que disponían los polacos e incorporarla al equipo, Churchill impone el más absoluto de los secretos. Se trata de poder descodificar los mensaje secretos alemanes; pero los alemanes no deben saber nunca que los ingleses conocen tal secreto.

La primitiva máquina Enigma es una máquina con tres rotores de giro independiente asociados a un teclado. Las diversas combinaciones de los rotores permiten biyecciones sofisticadas entre los conjuntos de letras del teclado. Así, poseyendo una clave, normalmente asociada al día, para ajustar los rotores, se puede transmitir información confidencial por medio de la radio. Los efectos del trabajo de Turing, sobre la base del trabajo preliminar, fueron esenciales en la Batalla de Inglaterra. Churchill y la fuerza aérea británica, eran capaces de predecir los movimientos de los grupos de bombarderos de la Luftwafe, consiguiendo, en muchos casos, interceptarlos. Aunque, a fuer de sinceros, hay que darle también al radar su papel en estos eventos.

Estos éxitos iniciales, hicieron que Churchill aumentara las dotaciones del proyecto "Ultra", creando distintos departamentos en constante ampliación. Dos elementos eran cruciales: el secretismo de sus trabajos no debía llegar a manos alemanas; pero ni siquiera los aliados deberían saber que disponían de medios para descodificar las máquinas Enigma alemanas.

Sin sospechar que todas, o muchas, de sus conversaciones estaban siendo escuchadas y transcritas, el ejército y la armada alemanes aumentaron el número de rotores por razones puramente instintivas. A. Turing también fue capaz de descodificar la nueva máquina Enigma usando simplemente lápiz y papel.

A mediados de 1942, los alemanes introducen una sofisticación adicional a sus comunicaciones por radio. Se trata del codificador de Lorentz de 12 rotores. Ahora, el número de posibles biyecciones entre teclados ha aumentado considerablemente. La nueva máquina, basada en el mismo principio, se incorpora en los submarinos alemanes.

Los británicos descubren bien pronto que los alemanes han cambiado su sistema criptográfico y es entonces cuando, por vez primera, observan la imposibilidad de seguir descodificando a mano. Apoyados por un alto presupuesto, por la voluntad

explícita de Churchill que considera su proyecto "Ultra" como la clave de la guerra, un grupo de ingenieros, con la colaboración de A. Turing, construye el primer ordenador electrónico de la historia. Se trata del ordenador *Colossus* y su hermano mayor *Colossus 2* que entraron en servicio en 1943 y estuvieron trabajando hasta el final de la Segunda Guerra Mundial. Ambos ordenadores eran capaces de procesar las combinaciones de la máquina de Lorentz y descodificar los mensajes de radio alemanes.

Cuando los norteamericanos entran en la Segunda Guerra Mundial, Churchill mantiene el secreto de su conocimiento del sistema criptográfico alemán. Sólo tras la Cumbre de Yalta, Churchill contará a Roosevelt que conoce el secreto; le transmitirá información ya descodificada; pero no le mostrará la existencia de las máquinas Colossus. En cuanto a Stalin, Churchill le transmitirá información; pero nunca llegará a informarle ni de la existencia del proyecto "Ultra" ni, mucho menos, de su funcionamiento. Sorprendentemente, Stalin ha conseguido infiltrar un hombre de su confianza en los barracones de Bletchley Park. Así Stalin conocerá todo el funcionamiento y evolución del proyecto "Ultra" sabiendo, al mismo tiempo, que sus aliados le mantienen apartado del secreto.

Con el final de la Segunda Guerra Mundial, y el advenimiento de la Guerra Fría, Churchill da órdenes de desmantelar el proyecto "Ultra", destruir los ordenadores Colossus y dispersar a los miembros de los equipos con la orden de guardar el secreto más absoluto. Así desaparecieron los primeros ordenadores electrónicos y su existencia no ha sido conocida hasta pasados los cincuenta años preceptivos de los Secretos Británicos.

De vuelta a sus actividades académicas, no siempre muy satisfactorias por la discriminación e incomprensión de sus "colegas", A. Turing participará en la creación de los ordenadores británicos Mark I y Mark II, ya metidos en la década de los cincuenta. Pensar en A. Turing reconstruyendo su modelo, casi diez años después de haberlo construido una vez, sólo por razones políticas, define un drama intelectual.

En los Estados Unidos, J. von Neumann, que ha dedicado bastante tiempo a la Teoría de Autómatas y, por ende, conoce bien la obra de Turing, es nombrado consejero matemático en la construcción de los primeros ENIAC estadounidenses, proto–ordenadores pensados para el desarrollo de tablas de trayectorias balísticas.

Desde entonces, hasta nuestros días, todos los ordenadores han mantenido las pautas de la máquina abstracta de Turing. En ocasiones, el modelo es modificado ligeramente para crear nuevas "Arquitecturas de Ordenadores", pero manteniendo siempre el concepto inicial de Turing: es el impopular lenguaje "ensamblador" que subyace a todos los ordenadores que conocemos.

Se produce un hecho extraordinario en la Historia de la Matemática: *Por vez primera un modelo teórico antecede al modelo físico, por vez primera en la Historia no hay que crear un modelo matemático de la realidad: es la realidad la que imita al modelo matemático.* Las consecuencias de tal fenómeno son, obviamente, extraordinarias para la posición de un matemático.

Por avanzar un poco en la historia, apresuradamente, concluyamos que, a partir de los trabajos de J. Robinson, el joven Ju. V. Matijasevic concluye en 1970 la

resolución del Problema X, demostrando la equivalencia entre los conjuntos recursivamente enumerables y los $\mathbb{Q}-$definibles.[24] Esto se traduce diciendo que no puede existir ningún algoritmo que resuelva el Problema X de Hilbert.

3. Máquinas de Turing

De los modelos de algoritmo que se encuentran involucrados en la definición de la Tesis de Church, dos de ellos han pervivido de manera notable.

- Las Funciones Recursivas: Constituyen el tipo de funciones a las que hay que poner mayor atención en Lógica y Teoría de Modelos.
- La Máquina de Turing: Además de su papel en Lógica y Teoría de Modelos, tienen un papel fundamental en Informática. Todo ordenador físico se ve modelizado por una máquina de Turing. Es uno de esos casos extraordinarios en los que el modelo matemático está perfectamente reflejado por la situación real (práctica) que modeliza.

Si la máquina de Turing refleja fielmente los ordenadores físicos, se convierte así en el Patrón, en la **unidad de medida** de todos los fenómenos observables físicamente en un ordenador. Así, la operación básica de una máquina de Turing (*la operación bit*) es la unidad de medida de la velocidad, normalmente medida en términos de *Mips:= millones de operaciones bit por segundo.*[25] La unidad de medida de capacidad de los ordenadores que compramos es la celda (el bit) del la máquina de Turing.[26] Por supuesto, al ser usado como Patrón y Unidad de Medida, *la máquina de Turing es el entorno natural matemático para el diseño y análisis de los algoritmos.* Así se muestra clásicamente en los textos básicos de Fundamentos de Informática Teórica.

3.0.2. Algunas Referencias Básicas del Texto. Entre las referencias básicas para estas notas destaquemos, antes de nada, la fuerte influencia de clásicos como [Ah-Ho-Ul, 75] (su nueva edición referenciada como [Ah-Ul, 95]) y la muy influyente serie [Kn, 68–05]. Sin embargo, deben destacarse como los textos más influyentes en la elaboración de estas páginas los dos volúmenes [Ba-Dí-Ga, 88] y [Ba-Dí-Ga, 90], así como el excelente texto [Pap, 94] y la excelente aparición del texto [Ar-Ba, 09]. Textos de gran calidad pueden ser "vieja biblia" [Wa-We, 86] (texto de lectura intratable, incluso para los especialistas, aunque, al mismo tiempo, debe destacarse como uno de los textos más completos del conocimiento de la época). Un pequeño texto, muy agradable para quienes provienen de un entorno matemático, son las notas para un curso semestral redactadas por D. Kozen y publicadas bajo la forma [Ko, 92]. Finalmente, no puedo menos que citar la guía de la intratabilidad [Ga-Jo, 79] que es, sin ninguna duda, uno de los textos que mejor ha sabido impulsar la investigación en Complejidad Computacional. Otros textos,

[24] Ju. V. Matijasevic. "Enumerable sets are definible". *Soviet Math. Dokl.* **11.2** (1970) 354–358.

[25] En ocasiones encontraremos la medida equivalente dada por el Mflop:= millones de operaciones coma flotante por segundo, que es obviamente equivalente a la de Turing.

[26] Con el tiempo se ha fijado como unidad el *byte* que equivale a 8 bits y, sucesivamente, el Kilobyte, el Megabyte, el Gigabyte y el Terabyte.

de menos agradable lectura, pero excelente reputación son los textos [Go, 99] y [Go, 08].

3.1. `La Noción: Máquinas de Turing.` Trataremos de fijar la noción de máquina de Turing. Como ya se señaló, el comportamiento de los ordenadores físicos en términos de complejidad (tanto tiempo como espacio) sigue las pautas del modelo teórico de las máquinas de Turing, de otro lado la arquitectura de los ordenadores reales mantiene la estructura interna de las máquinas de Turing o de variantes suyas.

Definición 3.1. *Llamaremos máquina de Turing (una sola cinta de Input (IT.) en la que autorizamos solamente lectura, k cintas de trabajo (WT.)) a todo quíntuplo* $M := (\Sigma, Q, q_0, F, \delta)$ *donde*

(1). Σ *es un conjunto finito (alfabeto),*
(2). Q *es un conjunto finito (espacio de estados),*
(3). $q_0 \in Q$ *es el estado inicial,*
(4). $F \subseteq Q$ *son los estados finales aceptadores,* $F_1 \subseteq Q$ *son los estados finales de rechazo.*
(5). *Una correspondencia (llamada función de transición)*

$$\delta : Q \times (\Sigma)^k \longrightarrow Q \times (\Sigma)^k \times \{-1, 0, 1\}^{k+1}.$$

Si δ *es una aplicación, la máquina de Turing* M *se denomina determinista, en caso contrario se denomina indeterminista.*[27]

Obviamente, no se puede entender el funcionamiento de una máquina de Turing sin entender su sistema de transición que presentaremos apoyándonos en su modelo gráfico.

3.2. `El Modelo Gráfico y el Sistema de Transición.` Introduciremos, para cada máquina de Turing

$$M = (\Sigma, Q, q_0, F, \delta),$$

un grafo orientado infinito $(S_M, \rightarrow_M)$, que denominaremos *sistema de transición* y que representa la acción (dinámica) de una máquina de Turing. Los elementos de S_M se denominan *configuraciones* (o *snapshots*) de la máquina M y representan la imagen de la máquina en un instante determinado. Las configuraciones vienen dadas por la siguiente definición:

$$S_M \subseteq Q \times (\Sigma^*)^{k+2} \times (\mathbb{N})^{k+2}$$

$$C := (q, x, y_1, \ldots, y_k, y_{k+1}, n_0, n_1, \ldots, n_k) \in Q \times (\Sigma^*)^{k+2} \times (\mathbb{N})^{k+2},$$

diremos que $C \in S_M$ si y solamente si se verifican las propiedades siguientes:

- $q \in Q$ es un estado (el estado de la configuración)
- $x := x_1, \ldots, x_n \in \Sigma^*$
- Para cada i, $1 \leq i \leq k$, se tiene

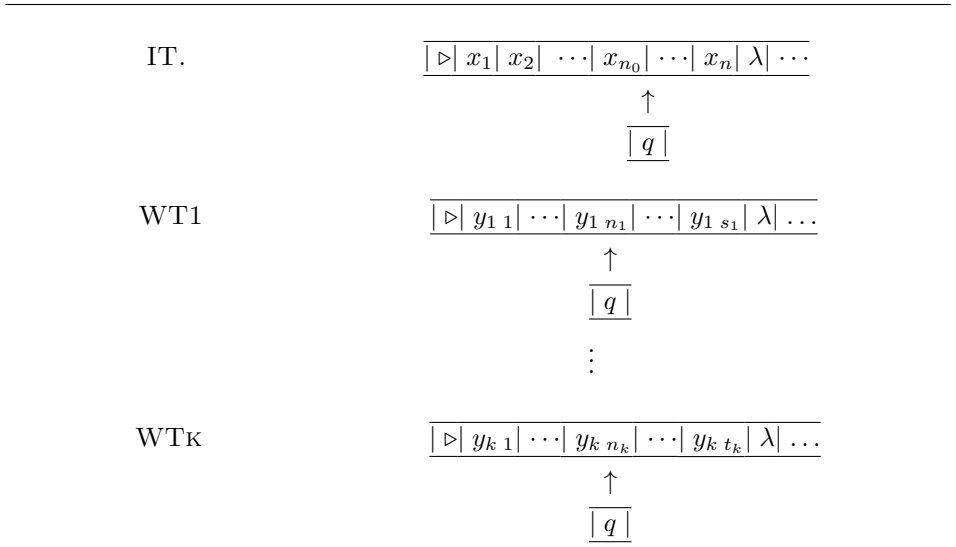
$$y_i := y_{i,1} \ldots y_{i,s_i} \in \Sigma^*$$

[27]Volveremos sobre el concepto de indeterminismo más adelante.

- $n_0, n_1, \ldots, n_k \in \mathbb{N}$ son las posiciones de las unidades de control, $0 \leq n_i \leq s_i + 1$, $1 \leq i \leq k$ y $0 \leq n_0 \leq n + 1$.

3.2.1. Modelo gráfico de una máquina de Turing. La *interpretación de una configuración* debe hacerse del modo siguiente. Utilizaremos un modelo gráfico que representa la dinámica de la máquina de Turing. Para ello, observaremos la siguiente descripción de una máquina de Turing:

Dispondremos de una cinta de input y k cintas de trabajo. Cada cinta está dividida en unidades de memoria (o celdas) que son capaces de contener un símbolo del alfabeto Σ (o el símbolo auxiliar $\triangleright$). Cada cinta tiene adosada una unidad de control, con una memoria finita. En esa unidad de control podemos guardar un estado (la idea es que las unidades de control tienen la capacidad de almacenar una cantidad finita de información). La configuración C anterior queda descrita mediante la figura siguiente:



Para poder interpretar la figura, debemos hacer las siguientes consideraciones:

- El estado q es el indicador de la fase de cálculo en la que nos encontramos. El estado q está guardado en las unidades de control (todas con el mismo estado, aunque en diferentes posiciones).
- Cada unidad de control está apuntando una celda de cada cinta. El número n_i representa el lugar al que está apuntando la unidad i. En realidad, estamos indicando el número del registro de memoria que debemos atender en la fase de LECTURA.
- Cada cinta actúa como un disco duro (o, si se prefiere, una partición del disco duro en $k + 1$ trozos). La información completa contenida en el disco duro no será utilizada simultáneamente. La cantidad de información utilizable es marcada por las unidades de control.

- El símbolo ▷ es el *cursor:* va a ser el representante del principio de cinta y sirve para prohibir (en la fase de Movimientos) el ir un paso a la izquierda del cursor. Lo que cuenta es la palabra que está descrita justo después.

3.2.2. Un paso de cálculo: Cuando una máquina de Turing se encuentra frente a una configuración como la descrita por la representación gráfica anterior, actúa del modo siguiente. Hay que dividir su acción en cinco etapas (el conjunto de todas ellas configura un *paso de cálculo* o, en términos técnicos, una *operación bit*).

(1). Parada.
(2). Lectura
(3). Transición
(4). Escritura
(5). Movimientos

`Parada.` Se trata de verificar la *Condición de Parada:* que viene marcada por los estados finales del modo siguiente:

---

while el estado q no es un estado final aceptador (i.e. $q \notin F$) **do**
Lectura
Transición
Escritura
Movimientos
od
Output el contenido de la cinta de trabajo $k-$ésima y termina la computación.
fi

Lectura. La fase de Lectura consiste en recuperar los contenidos de las celdas de las cintas señaladas por las unidades de control. Así, tras verificar la condición de parada, procedemos a "leer"los contenidos de las distintas cintas de trabajo:

Lectura: $= (q, x_{n_0}, y_{1,n_1}, \ldots, y_{k,n_k}) \in Q \times \Sigma^*$.
go to Transición

Transición. La máquina acude con el resultado de Lectura a la función de transición δ (o, para ser más precisos, al grafo de la función de transición). Se obtiene el resultado de la transición mediante:

Transición:$= \delta(\text{Lectura}) = (q', w_1, \ldots, w_k, \varepsilon_0, \ldots, \varepsilon_k) \in Q \times \Sigma^k \times \{-1, 0, 1\}^{k+1}$.
go to Escritura

Escritura. El proceso de escritura contiene dos etapas. La primera es cambiar el contenido de las unidades de control (esto es cambiar el estado q que estaba en las unidades de control por el nuevo estado q'). La segunda es cambiar el contenido de cada celda de cada cinta de trabajo (en el lugar donde estaba señalado) escribiendo el símbolo w_i. Esto se expresa diciendo:

Nuevo Estado: $q := q'$
for $i = 1$ to k **do**
    $y_{i,n_i} := w_i$
od
go to Movimientos

Movimientos. Ahora se trata de mover las unidades de control conforme a las reglas indicadas en la lista de movimientos

$$(\varepsilon_0, \ldots, \varepsilon_k) \in \{-1, 0, 1\}^{k+1},$$

conforme a las reglas siguientes:

$$\begin{array}{rcl} \varepsilon = -1 & \equiv & \text{Un paso a la izquierda} \\ \varepsilon = 0 & \equiv & \text{No te muevas} \\ \varepsilon = 1 & \equiv & \text{Un paso a la derecha} \end{array}$$

En otras palabras, la unidad de control se mueve a la celda inmediatamente a su izquierda si $\varepsilon = -1$, no se mueve si $\varepsilon = 0$ y se mueve a la celda inmediatamente a su derecha si $\varepsilon = 1$. Así, las posiciones de las unidades de control se modifican mediante el siguiente proceso:

---

for $i = 0$ **to** k **do**
    $n_i := n_i + \varepsilon_i$
od
go to Parada

El resultado de un paso (esto es, de las cuatro etapas descritas) es la configuración

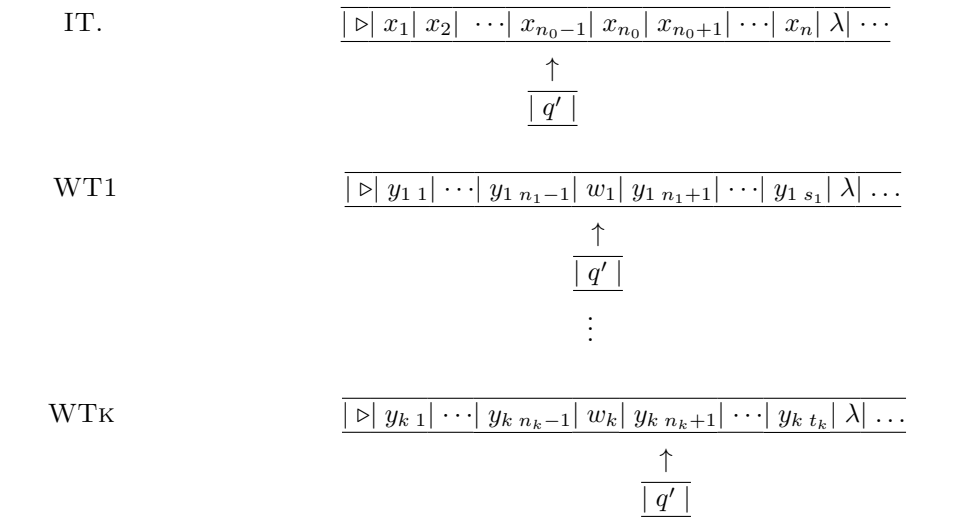
$$C' := (q', x, y'_1, \ldots, y'_k; n'_0, \ldots, n'_k),$$

donde

- q' es el nuevo estado,
- el input x no ha sido modificado,
- y'_i es como y_i salvo en el lugar n_i donde y_{i,n_i} ha sido reemplazado por w_i
- las nuevas posiciones han sido cambiadas de acuerdo a los movimientos, esto es,

$$n'_i := n_i + \varepsilon_i.$$

Gráficamente, la nueva configuración tendrá el siguiente aspecto, donde hemos supuesto que el $\varepsilon_0 = -1$, $\varepsilon_1 = 0$ y $\varepsilon_k = 1$.



Observación 3.2. Un ejercicio para entender cómo funciona una máquina de Turing consiste en definir las máquinas de Turing asociadas a las operaciones elementales de la aritmética (en base 2 para simplificar las tablas de cálculo) al modo escolar: Suma de naturales, Producto de naturales, División euclídea, etc.

3.3. `Algoritmos, funciones computables. Indecidibilidad.` Dadas dos configuraciones C y C' de una máquina de Turing M, escribiremos $C \to_M C'$ para denotar que C' se obtiene desde C en *un paso de cálculo* (un paso de deducción o una operación bit). Escribiremos $C \vdash_M C'$ para denotar que C' se puede obtener de la configuración C en un número finito de pasos de cálculo de la máquina de Turing M (es decir, *alcanzable por un camino finito dentro del grafo del sistema de transición asociado*).

Definición 3.3 (Tesis de Church)**.** *Se llama* **algoritmo** *o* **programa** *a toda máquina de Turing.*

Definición 3.4 (Terminología Básica)**.** *Dada una palabra $x \in \Sigma^*$ llamaremos* **configuración inicial sobre** *x a la configuración*

$$I(x) := (q_0, \triangleright x, \triangleright, \ldots, \triangleright, 1, \ldots, 1) \in S_M,$$

esto es, todas las cintas de trabajo están vacías salvo la cinta de Input donde aparece la palabra x. Las unidades de control están sobre el cursor para empezar a trabajar.

Una palabra $x \in \Sigma^*$ se dice **aceptada** *por una máquina de Turing M si a partir de la configuración inicial $I(x)$ se alcanza una configuración C (i.e. $I(x) \vdash_M C$*

en la que el estado es un estado final aceptador (i.e. el estado de C está en F). El conjunto de las palabras aceptadas por M se llama **lenguaje aceptado por** M *y es un subconjunto de Σ^* que se denota por $L(M)$.*

Si $x \in L(M)$, existe C configuración final aceptadora tal que $I(x) \vdash_M C$. En ese caso, el output es el contenido de la $k-$ésima cinta de trabajo y se dice que

$$Res_M(x) := y_k \in \Sigma^*,$$

es el **resultado (output)** *de la máquina de Turing sobre el input aceptado x.*

Definición 3.5. *Un lenguaje $L \subseteq \Sigma^*$ se llama recursivamente enumerable si es el lenguaje aceptado por alguna máquina de Turing (i.e. $L = L(M)$ para alguna máquina M). Se llama recursivo si tanto L como su complementario $\Sigma^* \setminus L$ son recursivamente enumerables.*

Definición 3.6 (Funciones Computables)**.** *Una función computable es una función*

$$f \ : D(f) \subseteq \Sigma^* \longrightarrow \Sigma^*,$$

tal que existe una máquina de Turing M sobre Σ tal que:

(1). $L(M) = D(f)$,
(2). $Res_M = f$.

3.3.1. La máquina Universal y el Problema de Parada. En su trabajo de 1936, A. Turing introdujo un ejemplo de problema recursivamente enumerable que no es recursivo. Ya se conocían los resultados de K. Gödel y A. Church; pero resulta interesante señalarlo aquí. Un problema recursivamente enumerable que no es recursivo es un *problema que se puede enunciar, pero no se puede resolver por métodos algorítmicos.* Dado que el ejemplo es notable, insistiremos en enunciarlo del modo siguiente:

Teorema 3.7 (A. Turing, 1936)**.** *Existe una máquina de Turing $\mathcal{U}$ sobre el alfabeto $\{0,1\}$ de tal modo que el lenguaje aceptado es dado por la siguiente definición:*

$$L(\mathcal{U}) := \Big\{(c_M, x) \ : \ \textit{tales que:}$$

- *c_M es el código binario de una máquina de Turing M,*
- *x es una palabra sobre el alfabeto de M tal que x es aceptada por M (i.e. $x \in L(M)$),*
- *el resultado de $\mathcal{U}$ sobre (c_M, x) es el mismo que el resultado de M sobre x, i.e.*

$$Res_{\mathcal{U}}(c_m, x) = Res_M(x), \forall x \in L(M)\Big\}.$$

La máquina $\mathcal{U}$ se denomina la **Máquina de Turing Universal**.

Junto a esta máquina Universal, A. Turing presentó el siguiente enunciado:

Teorema 3.8 (Problema de Parada)**.** *El siguiente lenguaje $HP \subseteq \{0,1\}^*$ es un lenguaje recursivamente enumerable que no es recursivo:*

$$HP := \{(c_M, x) \ : \ x \in L(M)\}.$$

La interpretación de estos dos resultados es la siguiente. En primer lugar, la máquina de Turing universal es también el lenguaje al que se transfiere (técnicamente compila, interpreta) todo programa, escrito en algún lenguaje de programación, en cada máquina concreta. Se conoce como *Lenguaje Máquina* o ensamblador y es el lenguaje al que traducen los compiladores los programas escritos en lenguajes de nivel más alto, para obtener un código ejecutable. A modo de ejemplo, A. Schönhage y sus colaboradores diseñaron la plataforma **TP** (Turing Processor). Esta plataforma está basada en una arquitectura de máquina de Turing, y diseñada sobre el modelo de la Máquina Universal, con un lenguaje de programación bastante duro (es un lenguaje ensamblador que se denomina **ALTP**: Assembler Language for the Turing Processor). Es la plataforma más eficaz para la programación de algoritmos de Matemáticas. Desgraciadamente, dado que el interfaz es bastante poco agradable, las gentes del entorno matemático prefieren programar en plataformas menos eficaces, pero más confortables y más difundidas como Maple, Matlab, Mathematica, etc. Para una referencia adecuada sobre **TP** y para solicitar el CD (gratuito) de instalación véase el libro [Sc-Ve, 94].

El Problema de Parada no es sólo un ejemplo de problema irresoluble algorítmicamente, sino que demuestra, además, que el sueño de la verificación es imposible. El Teorema de A. Turing dice que no puede existir un verificador universal de programas, dando pie a la *Programación Estructurada.*

3.3.2. Sobre Números Reales Recursivamente Enumerables. La relevancia del Problema de Parada es mucho mayor que la que simplemente aparece en el contexto de la informática. Se trata de un ejemplo de problema insoluble algorítmicamente sobre el que reposan muchos otros. Así, uno podría reconstruir el Teorema de Indecidibilidad de Gödel mediante una reducción al Problema de Parada. También la insolubilidad algorítmica de los problemas de Palabra en Semi–grupos y Grupos se reducen de manera natural al Problema de Parada. Sin embargo, Turing se proponía estudiar lo definible en el cuerpo de los números reales $\mathbb{R}$, para la fundamentación del Análisis. Usando un lenguaje más moderno, veamos qué números reales son definibles:

Introduzcamos una notación. Dado el alfabeto binario $\Sigma_0 := \{0, 1\}$ y dada una palabra

$$x := x_1 \dots x_n \in \Sigma_0^*,$$

denotaremos por $i(x)$ el número natural correspondiente, esto es,

$$i(x) := x_1 + 2x_2 + \cdots + 2^{n-1}x_n.$$

Definición 3.9. *Un número real $\alpha \in \mathbb{R}$ se dice recursivamente enumerable si existe una máquina de Turing M sobre el alfabeto Σ_0 que acepta un lenguaje $L(M) \subseteq \Sigma_0^*$ y evalúa una función recursivamente enumerable: $Res_M : L(M) \subseteq \Sigma_0^* \longrightarrow \Sigma_0^*$, tal que para todo $x \in \Sigma_0^*$ se tiene: $i(Res_M(x)) \in \{0, 1, 2, \dots, 9\}$, de tal modo que existe un número entero $a \in \mathbb{Z}$ verificándose la siguiente igualdad:*

$$\alpha = a + \sum_{x \in L(M)} \frac{i(Res_M(x))}{10^{i(x)}}.$$

Denotemos por $\mathbb{R}_{re} \subseteq \mathbb{R}$ el conjunto de los números reales recursivamente enumerables.

Llamaremos código de un número real recursivamente enumerable α al par $(a, M) \in \Sigma_1^*$ formado por su parte entera y por la máquina de Turing M que describe la expansión decimal de su mantisa.

(1). Obsérvese que el conjunto de los números reales recursivamente enumerables es un conjunto contable, luego es un subconjunto propiamente contenido en el cuerpo de los números reales.
(2). La idea de número real recursivamente enumerable se basa en un principio elemental. De una parte, su parte entera es dada, de otra parte la expansión decimal del número es calculable por una máquina de Turing.
(3). Obsérvese que los números racionales y los números reales algebraicos son recursivamente enumerables (los segundos mediante el método de Newton, por ejemplo).
(4). Obsérvese que $\mathbb{R}_{re}$ es un cuerpo realmente cerrado.
(5). Algunos números trascendentes son recursivamente enumerables como, por ejemplo, los números π y e, pues sus expansiones decimales se pueden calcular siempre usando, respectivamente, las expresiones de Stirling y de Neper. También son r.e. los números de Liouville, esto es, los números del tipo
$$\sum_{k=1}^{\infty} \frac{1}{a^{k!}},$$
donde $a \in \mathbb{N} \setminus \{0\}$. Los números de Lindemann (i.e. los números trascendentes de la forma e^{α}, donde α es un número algebraico irracional) también son recursivamente enumerables.
(6). Obsérvese que la definición de recursivamente enumerables indica esencialmente que se trata de números que podemos dar a alguien para que haga algo con ellos (i.e. son definibles con una cantidad finita de información).

Y, sin embargo, no es mucho lo que se puede hacer algorítmicamente con los números reales r.e.

Teorema 3.10. *El siguiente problema no es recursivo:*

$$Ineq := \{x, y \in \mathbb{R}_{re}^2 \ : \ x \geq y\}.$$

En otras palabras, no existe ningún algoritmo que decida para dos números reales dados cuál es el mayor entre ellos.

La demostración se sigue mostrando que todo algoritmo que resuelva *Ineq* resuelve también el Problema de Parada.

4. Funciones y Clases de Complejidad

La utilización de las máquinas de Turing para el análisis de la complejidad de algoritmos se remonta a los años 60. Entre los trabajos iniciales para modelizar

el fenómeno de la complejidad computacional cabe destacar los trabajos de M. Blum ([Bl, 67]), J. Hartmanis y R. Stearns ([Ha-St, 65]), M.O. Rabin ([Ra, 60, Ra, 66]). Son Hartmanis y Stearns quienes inician la ideología de las funciones de tiempo y espacio como funciones del tamaño de la entrada en su trabajo de 1965. Casi en la misma época, se establece la **Tesis de Cobham–Edmonds** (cf. [Co, 65], [Ed, 65a]) sobre los problemas *Tratables* informáticamente. Salvo la potencial aparición de la computación cuántica, el modelo de máquina de Turing aún permanece como modelo de complejidad razonable. Las nociones básicas son las siguientes:

Definición 4.1. *Sea M una máquina de Turing sobre el alfabeto Σ.*

(1). *Para $x \in \Sigma^*$, una palabra aceptada por la máquina M (i.e. $x \in L(M) \subseteq \Sigma^*$) llamaremos tiempo de la máquina M sobre el input x al número de operaciones bit (pasos de cálculo) que la máquina M realiza cobre la configuración inicial en x (i.e. $I(x)$) hasta que alcanza alguna configuración final aceptadora. Se denota tal función por $T_M(x) \in \mathbb{N}$.*

(2). *Para una palabra $x \in \Sigma^*$ se denomina talla de x (y se denota mediante $|x| \in \mathbb{N}$) al número de símbolos de Σ que contiene la palabra x.*

(3). *Dada una configuración C de M, se llama talla de la configuración (y se denota por $|C|$) al número total de bits ocupados en las distintas cintas de trabajo. En otras palabras, es el número de celdas ocupadas por símbolos de Σ en la configuración C.*

(4). *Dada una palabra $x \in L(M) \subseteq \Sigma^*$, llamaremos espacio–memoria consumido por M sobre x al máximo de las tallas de las configuraciones intermedias que surgen en el cálculo de M que comienza en la configuración inicial en x ($I(x)$) y termina en alguna configuración final aceptadora C. Se denota esta cantidad mediante $S_M(x) \in \mathbb{N}$.*

Observación 4.2. Dada la orientación de este mini–curso, es conveniente insistir en las frases "alguna configuración final aceptadora" usada en las definiciones de T_M y S_M. Dada una palabra $x \in \Sigma^*$ y dada la configuración inicial $I_M(x)$ podemos construir un grafo (potencialmente infinito) que tiene a $I_M(x)$ como raíz: se trata del subgrafo del sistema de transición asociado a M formado por todas las posibles configuraciones alcanzables desde $I_M(x)$. Este grafo es notablemente distinto en el caso determinista e indeterminista. En el primer caso, si $x \in L(M)$ el grafo sólo tiene un camino que comienza en $I_M(x)$ y termina en una (y sólo una) configuración final aceptadora posible (a partir de x). En el caso indeterminista puede haber varios posibles sucesores de cada nodo y por tanto un número potencialmente alto (posiblemente infinito) de caminos que comienzan en $I_M(x)$. En el caso indeterminista, para $x \in L(M)$, hay algún camino finito que alcanza alguna configuración final. Para contabilizar el tiempo y/o el espacio buscaremos el mejor camino posible entre aquellos que terminan en una configuración final aceptadora.

A partir de las funciones T_M y S_M se definen dos tipos de funciones de complejidad.

Definición 4.3 (Complejidad del Caso Peor)**.** *Sea M una máquina de Turing sobre el alfabeto Σ. Definimos las funciones siguientes:*

(1). *Función de Tiempo: $T_M : \mathbb{N} \longrightarrow \mathbb{R}_+$ dada mediante:*

$$T_M(n) := max\{T_M(x) \ : \ x \in L(M), \ |x| \leq n\},$$

es el "peor" de los tiempos de todos los inputs representables con, a lo sumo, n bits.

(2). *Función de Espacio: $s_M : \mathbb{N} \longrightarrow \mathbb{R}_+$ dada mediante:*

$$S_M(n) := max\{S_M(x) \ : \ x \in L(M), \ |x| \leq n\},$$

es el "peor" de los espacios de todos los inputs representables con, a lo sumo, n bits.

Observación 4.4. En ocasiones se utilizan las funciones de complejidad en promedio para el análisis del comportamiento de algoritmos; pero no incluiremos aquí esa discusión.

Los primeros resultados naturales (argumentos de diagonalización) se pueden resumir en el siguiente Teorema (cf [He-St, 66] y [Ha-Le-St, 65]).

Teorema 4.5. *Existe una máquina de Turing universal $\mathcal{U}$ verificando las propiedades de la máquina descrita en la Subsección 3.3.1 anterior, y tal que*

$$T_{\mathcal{U}}(M, x) \leq C_M T_M(x) \log_2 T_M(x),$$

y

$$S_{\mathcal{U}}(M, x) \leq D_M S_M(x),$$

donde C_M y D_M son dos constantes que dependen solamente del tamaño del espacio de estados de M, el tamaño del alfabeto y del número de cintas de trabajo.

Para evitar discusiones técnicas que nos alejen de nuestros objetivos, haremos una pequeña restricción: nos restringiremos a máquinas cuyas funciones de complejidad están acotadas por funciones constructibles en tiempo o espacio.

Definición 4.6 (Funciones Constructibles en Tiempo y/o en Espacio)**.** *Sea $f : \mathbb{N} \longrightarrow \mathbb{N}$ una función monótona.*

(1). *Diremos que f es constructible en tiempo si existe una máquina de Turing determinista M sobre el alfabeto unario $\Sigma := \{1\}$ tal que M termina su computación en todos los inputs y tal que existe una constante $c > 0$ verificando que para todo $n \in \{1\}^* = \mathbb{N}$, la máquina M calcula $f(n) \in \{1\}^* = \mathbb{N}$ en tiempo $t_M(n) \leq cf(n)$.*

(2). *Diremos que f es constructible en espacio si existe una máquina de Turing determinista M sobre el alfabeto unario $\Sigma := \{1\}$ tal que M termina su computación en todos los inputs y tal que existe una constante $c > 0$ verificando que para todo $n \in \{1\}^* = \mathbb{N}$, la máquina M calcula $f(n) \in \{1\}^* = \mathbb{N}$ en espacio $s_M(n) \leq cf(n)$.*

Ejemplo 4.7. (1). *Son funciones constructibles en tiempo las siguientes:*

- *$t(n) := n^k$, para $k \in \mathbb{N}$ fijo,*

- $t(n) := n!$,
- $t(n) := 2^{c.n}$, *para* c *fijo,*
- $t(n) := 2^{n^k}$, *para* $k \in \mathbb{N}$ *fijo,*
- $t(n) := n\lfloor \log_2 n \rfloor^k$, *para* $k \in \mathbb{N}$ *fijo,*
- $t(n) := 2^{2^{c.n}}$, *para* $c > 0$ *fijo.*

(2). *Son funciones constructibles en espacio las siguientes:*

- *Todas las constructibles en tiempo y, por ejemplo,*
- $t(n) := \lfloor \log_2 n \rfloor^k$, *para* $k \in \mathbb{N}$ *fijo,*

La restricción a cotas constructibles nos permite, por ejemplo, evitar la repetición de configuraciones (ciclos) mediante la introducción de contadores. Por ejemplo, nos restringiremos a situaciones como la descrita en la siguiente Proposición.

Proposición 4.8. *Sea* Σ *un alfabeto finito y* M *una máquina de Turing sobre* Σ. *Sea* $t : \mathbb{N} \longrightarrow \mathbb{N}$ *una función constructible en tiempo y supongamos* $t_M(n) \leq t(n)$ *para cada* $n \in \mathbb{N}$. *Sea* $L \subseteq \Sigma^*$ *el lenguaje aceptado por* M. *Entonces, existe una máquina de Turing* N *tal que se verifican las propiedades siguientes:*

- $L(N) := \Sigma^*$,
- $t_N(n) \in O(t)$,
- $Res_N(x) := \chi_L$, *donde:*

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

es la función característica de L.

- *El sistema de transición de* N *no repite configuraciones, esto es, para cada* $c \in S_N$ *no es cierto* $c \to_N c$.

Los siguientes resultados también pertenecen a la colaboración entre Hartmanis, Stearns, Lewis y Heine:

Proposición 4.9. *Los siguientes resultados muestran la independencia de las funciones de complejidad en términos de cambios de alfabeto y cambio del número de cintas.*

(1). Cambio de Alfabeto: *Sean* M *y* M_1 *dos máquinas de Turing que resuelven el mismo problema, mediante el mismo algoritmo en diferentes alfabetos* Σ *y* τ. *Supongamos que ambos alfabetos tienen al menos dos elementos. Entonces, las funciones de tiempo y espacio se relacionan mediante la siguiente expresión:*

$$T_{M_1}(n) \leq O(n + sT_M(\lfloor \frac{n}{s} \rfloor)),$$

y

$$s_{M_1}(n) \leq O(ss_M(\lfloor \frac{n}{s} \rfloor)),$$

donde $O(\cdot)$ *es la notación estándar del análisis y* $s := \log_2 \sharp(\tau)$.

(2). Cambio del número de cintas: *Sea* Σ *un alfabeto finito y sea* $M := (\Sigma, Q, q_o, F, \delta)$ *una máquina de Turing sobre el alfabeto* Σ. *Supongamos que* M *utiliza* k *cintas de trabajo. Entonces, existe una máquina de Turing*

M_1 sobre el mismo alfabeto con menos de 6 cintas de trabajo, y tal que se verifica $L(M_1) = L(M)$, $Res_{M_1} = Res_M$, y las funciones de tiempo y espacio mantienen la siguiente relación.

$$T_{M_1}(n) \leq O(T_M(n) \log_s T_M(n)),$$
$$S_{M_1}(n) \leq O(S_M(n)).$$

Estos resultados se complementan mediante el siguiente par de resultados debidos a Stearns y Hartmanis ([Ha-St, 65]).

Teorema 4.10 (Tape Compression Lemma)**.** *Sea $L \subseteq \Sigma^*$ un lenguaje aceptado por una máquina de Turing (determinística o no) usando espacio acotado por una función $s : \mathbb{N} \longrightarrow \mathbb{R}_+$ constructible en espacio, y sea $c \in \mathbb{R},\ 0 < c < 1$. Entonces, existe un alfabeto τ tal que $\Sigma \subseteq \tau$ y una máquina de Turing M_1 (del mismo tipo de determinismo) sobre el alfabeto τ tal que $L(M_c) = L$, $Res_{M_c} = Res_M$ y*

$$S_{M_c}(n) \leq cS_M(n).$$

En otras palabras, con un simple aumento del alfabeto podemos *siempre* obtener una disminución lineal de nuestros requisitos de memoria.

Teorema 4.11 (Linear Speed–Up)**.** *Sea $L \subseteq \Sigma^*$ un lenguaje aceptado por una máquina de Turing (determinística o no) usando tiempo acotado por una función $t : \mathbb{N} \longrightarrow \mathbb{R}_+$ constructible en tiempo y sea $c \in \mathbb{R},\ 0 < c < 1$. Entonces, existe un alfabeto τ tal que $\Sigma \subseteq \tau$ y una máquina de Turing (del mismo tipo de determinismo) sobre el alfabeto τ, tal que $L(M_c) = L$, $Res_{M_c} = Res_M$ y*

$$T_{M_c}(n) \leq 2n + cT_M(n).$$

Es decir, con un simple cambio de alfabeto también podemos conseguir acelerar linealmente cualquier proceso.

Ahora bien, la combinación de los resultados anteriores se traduce en la siguiente máxima:

Se puede acelerar linealmente la velocidad de cálculo o reducir linealmente los requerimientos de memoria de un proceso (basta con comprar otro ordenador más potente); pero de esa aceleración no se deduce que se pueda reducir el tipo asintótico de la función de complejidad considerada.

En otras palabras, salvo una constante, que se puede llamar la constante del ordenador, las funciones de complejidad en tiempo y espacio se clasifican solamente en términos de la clase asintótica, esto es, las clases de funciones $O(T_M)$ u $O(S_M)$. Esto justifica el estudio de clases de complejidad (y sus relaciones) atendiendo a la asintótica. Como el alfabeto no es relevante se suele fijar el alfabeto $\Sigma := \{0, 1\}$.

4.1. `Clases en Términos de Complejidad.` Los problemas a clasificar son esencialmente los Problemas Decisionales. Un Problema decisional está determinado por un lenguaje $L \subset \Sigma^*$ sobre un alfabeto finito. Se trata de evaluar la función característica definida por L, esto es, la función $\chi_L : \Sigma^* \longrightarrow \{0, 1\}$ dada mediante:

$$\chi_L(x) := \begin{cases} 1 & \text{si } x \in L \\ 0 & \text{en caso contrario.} \end{cases}$$

Los problemas decisionales se clasifican mediante:

Definición 4.12 (Clases Determinísticas)**.** *Se definen las siguientes clases de complejidad para los lenguajes recursivamente enumerables $L \subseteq \{0,1\}^*$. Sea $f : \mathbb{N} \longrightarrow \mathbb{R}_+$ una función monótona creciente. Se define:*

$$\mathbf{DTIME}(f) := \{L \subseteq \{0,1\}^* : \textit{existe una máq. de Turing determinística } M \textit{ t.q. } L = L(M) \textit{ y } T_M \in O(f)\}.$$
$$\mathbf{DSPACE}(f) := \{L \subseteq \{0,1\}^* : \textit{existe una máq. de Turing determinística } M \textit{ t.q. } L = L(M) \textit{ y } S_M \in O(f)\}.$$

Definición 4.13 (Clases Indeterministas)**.** *Se definen las siguientes clases de complejidad para los lenguajes recursivamente enumerables $L \subseteq \{0,1\}^*$. Sea $f : \mathbb{N} \longrightarrow \mathbb{R}_+$ una función monótona creciente. Se define:*

$$\mathbf{NTIME}(f) := \{L \subseteq \{0,1\}^* : \textit{existe una máq. de Turing indeterminística } M \textit{ t.q. } L = L(M) \textit{ y } T_M \in O(f)\}.$$
$$\mathbf{NSPACE}(f) := \{L \subseteq \{0,1\}^* : \textit{existe una máq. de Turing indeterminística } M \textit{ t.q. } L = L(M) \textit{ y } S_M \in O(f)\}.$$

Ya hemos justificado el uso del "comportamiento asintótico" $O(f)$ por los resultados de Stearns y Hartmanis antes expuestos. Inmediatamente tenemos las siguientes relaciones:

$$\mathbf{DTIME}(f) \subseteq \mathbf{NTIME}(f),$$
$$\mathbf{DTIME}(f) \subseteq \mathbf{DSPACE}(f) \subseteq \mathbf{NSPACE}(f).$$

Y si $f \in O(g)$, tendremos

$$\mathbf{DTIME}(f) \subseteq \mathbf{DTIME}(g),$$
$$\mathbf{NTIME}(f) \subseteq \mathbf{NTIME}(g),$$
$$\mathbf{DSPACE}(f) \subseteq \mathbf{DSPACE}(g),$$
$$\mathbf{NSPACE}(f) \subseteq \mathbf{NSPACE}(g).$$

Los Teoremas de Jerarquía de Tiempo y Espacio, debidos también a Stearns y Hartmanis, justifican el uso del término "clasificación".

Definición 4.14. *Sean $f, g : \mathbb{N} \longrightarrow \mathbb{R}_+$, dos funciones monótonas crecientes. Diremos que $g \in \omega(f)$ si para cada $c > 0$, existe un conjunto infinito $I_c \subseteq \mathbb{N}$, tal que*

$$g(n) > cf(n).$$

Teorema 4.15 (Teorema de Jerarquía en Espacio)**.** *Sean $s, s' : \mathbb{N} \longrightarrow \mathbb{N}$ dos funciones constructibles en espacio y supongamos que $s' \in \omega(s)$ (esto es, $s' \notin O(s)$). Entonces,*

$$\mathbf{DSPACE}(s') \setminus \mathbf{DSPACE}(s) \neq \emptyset,$$
$$\mathbf{NSPACE}(s') \setminus \mathbf{NSPACE}(s) \neq \emptyset.$$

Teorema 4.16 (Teorema de Jerarquía en Tiempo)**.** *Sean* $t, t' : \mathbb{N} \longrightarrow \mathbb{N}$ *dos funciones monótonas crecientes y supongamos que* $t' \in \omega(t \log t)$*, y* $n \in O(t')$*.*

Entonces,

$$\mathbf{DTIME}(t') \setminus \mathbf{DTIME}(t) \neq \emptyset.$$

En otras palabras, la clasificación anterior es justa. Uno puede encontrar problemas cuyo tiempo es una función polinomial $O(n^5)$ pero que no se resuelven en tiempo $O(n^4)$, por ejemplo.

Un resultado sorprendente de los primeros tiempos es el Teorema de aceleración de M. Blum en [Bl, 67]. Una "traducción" del enunciado de Blum es debida a Trakhtenbrot ([Tr, 64]) y A. Borodin [Bo, 72] y se conoce como el *Gap Theorem.*

Teorema 4.17 (Gap Theorem)**.** *Dada una función computable* $g : \mathbb{N} \longrightarrow \mathbb{N}$*, tal que* $g(n) \geq n$*, para cada* $n \in \mathbb{N}$*, entonces existe una cota de tiempo* T *tal que* $\mathbf{DTIME}(g \circ T) \subseteq \mathbf{DTIME}(T)$*.*

Este enunciado no contradice el Teorema de Jerarquía. Más aún, como la cota T puede ser muy grande, no aporta gran cosa a la relación entre **P** y **NP** que nos ocupa aquí.

En ocasiones, sin embargo, uno puede estar interesado en determinar la complejidad de una cierta función. En este caso se consideran las clases de complejidad de funciones. Notacionalmente, se añade el sufijo **F** al nombre de la clase:

Definición 4.18 (Clases de complejidad de funciones)**.** *Sea* $f : \mathbb{N} \longrightarrow \mathbb{R}_+$ *una función monótona creciente y sea* $\varphi : D(\varphi) \subseteq \Sigma^* \longrightarrow \Sigma^*$ *una función.*

(1). *Decimos que* $\varphi \in \mathbf{DTIMEF}(f)$ *si existe una máquina de Turing determinística* M *tal que:*
   a) $L(M) := D(\varphi)$*,*
   b) $Res_M(x) := \varphi(x)$*,*
   c) $T_M(n) \in O(f)$*.*

(2). *Decimos que* $\varphi \in \mathbf{DSPACEF}(f)$ *si existe una máquina de Turing determinística* M *tal que:*
   a) $L(M) := D(\varphi)$*,*
   b) $Res_M(x) := \varphi(x)$*,*
   c) $S_M(n) \in O(f)$*.*

4.2. `Rudimentos con Indeterminismo.` Algunos resultados primarios en el manejo del indeterminismo se pueden mostrar en los siguientes resultados. Recuérdese que el indeterminismo se puede representar mediante la presencia de grafos en el espacio de configuraciones. Por tanto, una posible manera de atacar determinísticamente problemas modelizados mediante máquinas indeterministas se basa en la generación de los grafos y en la reducción a un problema de **Alcanzabilidad**. Este es el fundamento de los resultados siguientes:

Teorema 4.19. *Sea* $t : \mathbb{N} \longrightarrow \mathbb{N}$ *una función constructible en tiempo,* $t(n) \geq n$*. Entonces,*

$$\mathbf{NSPACE}(t) \subseteq \mathbf{DTIME}(2^{O(t)}).$$

Se basa en la simple construcción para cada $x \in \Sigma^*$ del grafo $\mathcal{G}_x$ cuyos vértices son todas las configuraciones de talla acotada por $t(|x|)$ y del cálculo de la clausura transitiva de la configuración inicial en x, $I_M(x)$, dentro de ese grafo.

El mismo grafo nos permitiría probar que $\mathbf{NTIME}(t) \subseteq \mathbf{DSPACE}(t^2)$, pero un poco de sutileza (describiendo las posibles transiciones, en lugar de los nodos de $\mathcal{G}_x$) nos permite refinar hasta:

Teorema 4.20. *Sea* $t : \mathbb{N} \longrightarrow \mathbb{N}$ *una función constructible en tiempo,* $t(n) \geq n$. *Entonces,*

$$\mathbf{NTIME}(t) \subseteq \mathbf{DSPACE}(t).$$

Un resultado bastante más sutil nos permite mostrar la excelente relación que tiene el espacio con el indeterminismo. Se trata de recorrer el grafo con estrategia *depth–first–search.*

Teorema 4.21. [Sa, 70] *Si* $s : \mathbb{N} \longrightarrow \mathbb{N}$ *es una función constructible en espacio y* $s(n) \geq \log_2 n$, *se tiene:*

$$\mathbf{NSPACE}(s) \subseteq \mathbf{DSPACE}(s^2).$$

Observación 4.22. Este resultado supone la primera observación relevante en relación con el objetivo del mini–curso: el indeterminismo es un concepto esencialmente irrelevante cuando se trata de enfrentar problemas de espacio/memoria. La conjetura de Cook pregunta, justamente, por la interacción entre indeterminismo y tiempo.

5. Clases Centrales de Complejidad

5.1. `La tesis de Cobham-Edmonds.` La tesis de Cobham–Edmonds se puede resumir en los términos siguientes:

Definición 5.1 (Cobham–Edmond's Thesis)**.** *Un problema algorítmico se debe considerar tratable si la complejidad en tiempo es polinomial en la talla del input.*

El crecimiento polinomial del tiempo parece una apuesta razonable [28] como definición. Sin embargo, complejidades, supuestamente tratables, en tiempo del orden de $O(n^{100})$ podrían ser consideradas como intratables en términos prácticos. Curiosamente se conocen problemas naturales con complejidades de orden exponencial o, incluso doblemente exponencial (véase el caso de la cota inferior $2^{2^{O(n)}}$ de E. Mayr y A. Meyer para el cálculo de bases de Gröbner). En cambio, se conocen pocos problemas "naturales" en los que, una vez demostrada su tratabilidad, se haya demostrado también que es imprescindible tener grados altos.[29]

[28] Frente a complejidades de orden exponencial o doblemente exponencial.

[29] Aunque siempre es discutible el término "natural". Un problema suele llamarse natural en la medida en que ha sido motivado por "agentes/causas externos" al mundo de la complejidad. Pero la relevancia de la "naturalidad" es cuestión de gusto y orientación personal. No es el caso de quien suscribe.

Definición 5.2 (La clase **P**)**.** *Se define la clase de los lenguajes tratables como la clase*

$$\mathbf{P} := \bigcup_{k\in\mathbb{N}} \mathbf{DTIME}(n^k).$$

Se define la clase de las funciones tratables **PF** *como la clase de funciones parciales* $f : \Sigma^* \longrightarrow \Sigma^*$ *evaluables en tiempo polinomial.*

5.2. `Centrales: Primeras Relaciones.`

(1). Clases Determinadas por el Tiempo: Se consideran clases deterministas e indeterministas. El prefijo **N** las distingue.

$$\mathbf{NP} := \bigcup_{k\in\mathbb{N}} \mathbf{NTIME}(n^k).$$
$$\mathbf{EXTIME} := \mathbf{DTIME}(2^{O(n)}).$$
$$\mathbf{NEXTIME} := \mathbf{NTIME}(2^{O(n)}).$$
$$\mathbf{EXPTIME} := \bigcup_{k\in\mathbb{N}} \mathbf{DTIME}(2^{n^k}).$$
$$\mathbf{NEXPTIME} := \bigcup_{k\in\mathbb{N}} \mathbf{NTIME}(2^{n^k}).$$
$$\mathbf{DDEXTIME} := \mathbf{DTIME}(2^{2^{O(n)}}).$$

(2). Clases de Funciones/Correspondencias: Se denotan añadiendo el sufijo **F**: **PF**, **NPF**, **EXTIMEF**, etc.

(3). Las clases de tiempo presentan contenidos estrictos como consecuencia de los Teoremas de Jerarquía en Tiempo.

$$\mathbf{P} \subsetneq \mathbf{EXTIME} \subsetneq \mathbf{EXPTIME} \subsetneq \mathbf{DDEXTIME}.$$
$$\mathbf{NP} \subsetneq \mathbf{NEXTIME} \subsetneq \mathbf{NEXPTIME}.$$

Las clases por encima de la clase **P** se consideran ya clases **intratables**. Una discusión especial merece la clase **NP** que se analizará más adelante.

(4). Clases Determinadas por el Espacio. Con contenidos estrictos como consecuencia de los Teoremas de Jerarquía en Espacio. La clase límite en la que aún se puede esperar tratabilidad es la clase **PSPACE**. Por encima de ella, los problemas se consideran intratables:

$$\mathbf{LOG} := \mathbf{DSPACE}(\log n) \subseteq \mathbf{NLOG} := \mathbf{NSPACE}(\log n).$$
$$\mathbf{PLOG} := \bigcup_{k\in\mathbb{N}} \mathbf{DSPACE}(\log^k n).$$
$$\mathbf{PSPACE} := \bigcup_{k\in\mathbb{N}} \mathbf{DSPACE}(n^k).$$
$$\mathbf{EXSPACE} := \mathbf{DSPACE}(2^{O(n)}).$$
$$\mathbf{EXPSPACE} := \bigcup_{k\in\mathbb{N}} \mathbf{DSPACE}(2^{n^k}).$$

(5). Exceptuando el caso de **NLOG**, las clases de espacio indeterminista no se consideran, como consecuencia del Teorema de Savitch. Los siguientes contenidos son estrictos:

$$\mathbf{LOG} \subsetneq \mathbf{PLOG} \subsetneq \mathbf{PSPACE} \subsetneq \mathbf{EXSPACE} \subsetneq \mathbf{EXPSPACE}.$$

5.2.1. Algunas Relaciones Más. Usando los resultados preliminares sobre relaciones entre clases, uno puede obtener

Corolario 5.3. *Se dan las siguientes relaciones entre las clases anteriores:*

(1). $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$.
(2). $\mathbf{PLOG} \subsetneq \mathbf{EXTIME}$.
(3). $\mathbf{NLOG} \subseteq \mathbf{P}$.

Demostración.– Los contenidos de los apartados (1) y (2) se siguen de anteriores disquisiciones. El contenido estricto $\mathbf{PLOG} \subsetneq \mathbf{EXTIME}$ se sigue de los Teoremas de Jerarquía en tiempo, dado que $\mathbf{PLOG} \subseteq \mathbf{DTIME}(2^{\log^{O(1)}(n)}) \subsetneq \mathbf{EXTIME}$. En cuanto al apartado iii), baste observar la estrategia siguiente:

- Dado un lenguaje $L \in \mathbf{NLOG}$, el número total de configuraciones con input $x \in \Sigma^*$ está acotado por $2^{O(\log(n))} = n^{O(1)}$.
- Diseñemos un grafo orientado en el sistema de transición con todas esas configuraciones posibles y un algoritmo de ALCANZABILIDAD.
- Aceptamos si y solamente si se alcanza una configuración final aceptadora a partir de la configuración inicial en x.

□

5.3. `La clase` **NP**`: Reflexión, ejemplos.` Debemos dar la prioridad en la definición y primeras exploraciones de la clase **NP** a Cook ([Cook, 71]), Levin ([Lev, 73]) y Karp ([Karp, 72]), así como los primeros problemas **NP**–completos. Existen referencias anteriores de K. Gödel en las que ya se pre–anuncia la relevancia de la clase; pero dejamos a manuscritos históricos, como [Fo-Ho, 03] y sus referencias, los detalles y discusiones de anteriores períodos.

El concepto que hemos presentado del Indeterminismo se basa en el modelo de máquina: una mera diferencia entre correspondencia y aplicación, basta para diferenciarlos. Sin embargo, esta presentación es muy poco clarificadora de lo que significa el indeterminismo. En sentido informático simplista, el determinismo sería "lo programable", y el indeterminismo serían los programas mal diseñados en los que el programa no sabe dilucidar cuál es el "paso siguiente" de la computación. Esta interpretación simplista dista brutalmente de la relevancia del concepto. Aquí, vamos a discutir un poco más la noción.

Definición 5.4 (Proyección de un lenguaje)**.** *Dado un lenguaje $L \subseteq \Sigma^*$, llamaremos proyección del lenguaje L al lenguaje*

$$\pi(L) := \{x \in \Sigma^* \ : \ \exists y \in \Sigma^*, \ x \cdot y \in L\},$$

donde $\cdot$ es la adjunción de palabras.[30]

[30]La operación natural sobre Σ^* que le confiere estructura de monoide.

He elegido el término proyección porque refleja el lenguaje geométrico (la proyección de una subvariedad de $\mathbb{K}^n$ hacia un espacio ambiente de menor dimensión), en el lenguaje de la Teoría de Modelos hablaríamos de la presencia de un bloque de cuantificadores existenciales. En términos de lenguajes, $\pi(L)$ serían los prefijos de las palabras en L (sin limitaciones).

Los ejemplos más obvios podrían ser los siguientes. Consideremos la "variedad de incidencia" sobre $\mathbb{Q}$ dada por las ecuaciones polinomiales multivariadas con coeficientes racionales:

$$L := V^{(\mathbb{Q})} := \{(f, x) \in \mathbb{Q}[X_1, \ldots, X_n] \times \mathbb{Q}^n \ : \ f(x) = 0\}.$$

Es obviamente un lenguaje sobre el alfabeto $\{0, 1\}$ con, por ejemplo, codificación densa de los coeficientes. Es claramente un lenguaje recursivo. De hecho, usando codificación densa, se puede decidir en tiempo polinomial si un dato $\omega \in \{0, 1\}^*$ está en L o no (basta con confirmar que son polinomios y racionales y evaluar un polinomio en un punto).

Si ahora proyecto L, obtengo $\pi(L)$, el conjunto de polinomios con coeficientes enteros que poseen una solución diofántica: es decir, caemos en el Problema X de Hilbert (del que ya dijimos que era indecidible). Es decir, en ciertas teorías la eliminación de cuantificadores es algorítmicamente indecidible.

Añadamos a la variedad de incidencia una condición sobre la talla de los ceros:

$$L_1 := V_1^{\mathbb{Q}} := \{(f, x) \in V^{\mathbb{Q}} \ : \ ht(x) \leq \binom{\deg(f) + n}{n} \|f\|_\Delta\},$$

donde $\|f\|_\Delta$ es la norma (unitariamente invariante) de Bombieri de f y $ht(x)$ es la altura de Weil del punto racional (i.e. la talla bit de x).

En este caso la proyección $\pi(L_1)$ ya es tratable algorítmicamente: basta con probar con todos los posibles valores de x de altura acotada por la cantidad prescrita (que no es otra cosa que la esperanza de $|f|$ sobre la esfera unidad en $\mathbb{C}^n$).

La proyección es también modelizable mediante el indeterminismo. Para ello, redescribimos el indeterminismo mediante máquinas que admiten una especial operación de "guessing" (adivinando la buena respuesta). Así supongamos que tenemos una máquina de Turing determinista N que resuelve el problema L, entonces tendríamos una "máquina" que acepta indeterminísticamente $\pi(L)$ y que tendría la estructura siguiente:

INPUT: $x \in \Sigma^*$
guess indeterministically $y \in \Sigma^*$
Aplicar N sobre $x \cdot y \in \Sigma^*$.
if N acepta $x \cdot y$,
    OUTPUT x es aceptado y $Res_N(x \cdot y)$.
else OUTPUT No respondas
fi

Las llamaremos máquinas con guessing. De hecho se puede probar lo siguiente:

Proposición 5.5. *Sea $L \in \mathbf{NTIME}(s)$ un lenguaje aceptable en tiempo indeterminista acotado por una función constructible en tiempo s. Entonces, existe una máquina de Turing determinística N tal que*

$$L := \{x \in \Sigma^* \ : \ \exists y \in \Sigma^*,\ |y| \leq cs(|x|),\ x \cdot y \in L(N)\},$$

de tal modo que $T_N \in O(s)$.

Lo único que hay que "adivinar" es un cadena de transiciones de longitud s y verificar que son factibles.

En otras palabras los lenguajes en $\mathbf{NTIME}(s)$ son los prefijos de palabras x de algún lenguaje de $\mathbf{DTIME}(s)$ de tal modo que hay sufijos que las continúan (en la fibra $\pi^{-1}(x) \cap L(N)$) que tienen talla acotada por s.

En el caso que nos ocupa:

Definición 5.6. *Se define la clase* **NP** *del modo siguiente:*

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

Este mismo concepto se expresa mediante la siguiente caracterización de los lenguajes en la clase **NP**.

Proposición 5.7. *Sea Σ un alfabeto finito y $L \subseteq \Sigma^*$ un lenguaje. Entonces, $L \in$* **NP** *si y solamente si existen:*

- *Dos funciones polinomiales $p, q \ : \mathbb{N} \longrightarrow \mathbb{N}$.*
- *Una máquina de Turing determinística M.*
- *Un lenguaje $\tilde{L} \subseteq \Sigma^*$*

tales que:

(1). *El lenguaje aceptado por M es $\tilde{L}$ (i.e. $\tilde{L} = L_M$).*

(2). *El tiempo de ejecución de la máquina M está acotado por p, i.e.*

$$T_M(n) \leq p(n), \forall n \in \mathbb{N}.$$

(3). *El lenguaje L está caracterizado por la propiedad siguiente:*

$$\forall x \in \Sigma^*, \ \ con \ \ |x| = n \in \mathbb{N}, \ \ x \in L \Leftrightarrow \exists y \in \Sigma^*, \ \ [|y| \leq q(n)] \wedge [x \cdot y \in \tilde{L}].$$

Un ejemplo sencillo de problema en la clase **NP** es el siguiente:

Problema 5.8 (`SAT`). *Se define el lenguaje* `SAT` *como el conjunto de las fórmulas $\Phi(X_1, \ldots, X_n)$ del Cálculo Proposicional que son satisfactibles, es decir, tales que existe una asignación de verdad $x := (x_1, \ldots, x_n) \in \{0,1\}^n$ tal que $\Phi(x) = 1$. En otras palabras,*

$$\texttt{SAT} := \{\Phi \ : \ \exists x \in \{0,1\}^n,\ \Phi(x) = 1\}.$$

Es claro que para decidir si una fórmula del Cálculo Proposicional es satisfactible, basta con "adivinar" una asignación de verdad y evaluar la fórmula en esa asignación.

Una de las razones que resalta la relevancia de la relación entre las clases **P** y **NP** en la interacción entre determinismo, indeterminismo y tiempo es el siguiente

resultado basado en un argumento de *"padding"*. La siguiente proposición explicaría que la igualdad entre las clases centrales **P** y **NP** se extendería a casi todas las clases de tiempo determinístico e indeterminístico en una forma que traduciría a tiempo el Teorema 4.21 de Savitch sobre espacio.

Proposición 5.9. *Si* $\mathbf{P} = \mathbf{NP}$*, entonces, para toda función* f *constructible en tiempo se tiene:*

$$\mathbf{DTIME}(2^{O(f)}) = \mathbf{NTIME}(2^{O(f)}), \quad \mathbf{DTIME}(f^{O(1)}) = \mathbf{NTIME}(f^{O(1)}).$$

Demostración.– Se trata de un argumento de "padding". Sea $L \in \mathbf{NTIME}(2^{O(f)})$ y supongamos que existe una máquina de Turing indeterminística M y una constante positiva $c > 0$ tal que M acepta L en tiempo $2^{cf(n)}$. Ahora consideremos el siguiente lenguaje:

$$L' := \{x10^{2^{cf(|x|)}} \ : \ x \in L\}.$$

Existe una máquina de Turing indeterminística que acepta el lenguaje L' en tiempo lineal. La máquina comienza buscando el último 1 a la derecha de la palabra. Así, en tiempo lineal, extrae la subpalabra x. Seguidamente aplica la máquina M sobre x. Puesto que hemos supuesto **P**=**NP**, existirá una máquina de Turing determinística M' que, en tiempo **P**, acepta el lenguaje l'. Ahora adaptamos esa máquina M' para aceptar L del modo obvio: dada una entrada x, añadimos a x $10^{2^{cf(|x|)}}$ y procedemos aplicando M'. □

En ocasiones tiene interés tratar las clases de complejidad de los complementarios de lenguajes.

Definición 5.10 (co–**C**)**.** *Sea* **C** *una clase de complejidad, definimos la clase de lenguajes co–***C** *del modo siguiente:*

$$co - \mathbf{C} := \{L \subseteq \Sigma^* \ : \ L^c \in \mathbf{C}\},$$

donde $L^c := \Sigma^* \setminus L$ *denota complementario.*

Una clase de especial relevancia es la clase co–**NP** de complementarios de lenguajes en **NP**. Se observa claramente que $\mathbf{P} \subseteq \mathbf{NP} \cap$ co–**NP**. Se desconoce la relación entre ambas clases. Se sabe, por ejemplo, que si **NP**$\neq$co–**NP**, entonces **P**$\neq$**NP**, pero se desconoce el recíproco de esa pregunta. Detenernos así genera un problema abierto más, dentro de la conjetura de Cook:

Problema Abierto 5.11. *Decidir la respuesta a la siguiente pregunta:*

$$\mathbf{NP} \neq co - \mathbf{NP}?$$

Un ejemplo de problema en co–**NP** es el siguiente:

Problema 5.12 (`TAUT`)**.** *Se define el lenguaje* `TAUT` *como el conjunto de las fórmulas* $\Phi(X_1, \ldots, X_n)$ *del Cálculo Proposicional que son tautologías, es decir, tales que para toda asignación de verdad* $x := (x_1, \ldots, x_n) \in \{0,1\}^n$ *se verifica* $\Phi(x) = 1$*. En otras palabras,*

$$\texttt{TAUT} := \{\Phi \ : \ \forall x \in \{0,1\}^n, \ \Phi(x) = 1\}.$$

Proposición 5.13. *El lenguaje* `TAUT` *está en co–***NP***.*

5.4. `Algoritmos Probabilistas:` **BPP** *et al.* En realidad, la tesis de Cobham–Edmonds debe extenderse hasta la clase de algoritmos tratables que incorporan un ingrediente de aleatoriedad: los algoritmos probabilistas con tiempo polinomial. En la práctica son los algoritmos más utilizados, tienden a ser más eficientes que los deterministas conocidos para problemas análogos. Su buen comportamiento en la práctica y su robustez teórica los hace deseables y valiosos.

Históricamente, nacen con los primeros tests de primalidad en los trabajos de Solovay y Strassen (cf. [So-St, 77]) o Miller y Rabin (cf. [Mi, 76], [Ra, 80]). Después vinieron los tests de nulidad de polinomios dados por programas que los evalúan (*straight–line program*) como en los trabajos [Sc, 80], [Zi, 90] o en las versiones con conjuntos questores [He-Sc, 82], [Kr-Prd, 96] (cf. [Prd, 95] para un histórico del tema). Los primeros tendrán gran impacto en el diseño de protocolos criptográficos como RSA (cf. [Ri-Sh-Ad, 78]), los segundos en el diseño de algoritmos eficientes en Teoría de la Eliminación (cf. [Gi-He-Prd *et al.*, 97]) y en el Tratamiento del Nullstellensatz de Hilbert (cf. [Ha-Mo-Prd-So, 00], [Kr-Prd-So, 01]). Más recientemente, los algoritmos probabilistas servirán para resolver el Problema XVII de los propuestos por S. Smale para el siglo XXI (cf. [Be-Prd, 09a], [Be-Prd, 11] y/o el *survey* [Be-Prd, 09b], por ejemplo). Una monografía sobre algoritmos probabilistas o aleatorios es [Mo-Ra, 95].

A primera vista, los algoritmos probabilistas tienen un aire similar a los indeterministas: también disponemos de una etapa de "guessing" sobre un conjunto de elementos de longitud polinomial en el tamaño de la entrada. La diferencia estriba en que en el caso probabilista *disponemos de un control de la probabilidad de cometer errores*. A partir de un polinomio univariado p y de una máquina de Turing determinística N, podemos imaginar un modelo de máquina de la forma siguiente:

Input: $x \in \Sigma^*$
guess at random $y \in \Sigma^*$, $|y| \leq p(|x|)$
Aplicar N sobre $x \cdot y \in \Sigma^*$.
if N acepta $x \cdot y$,
    Output x es aceptado y $Res_N(x \cdot y)$.
else Output rechazar x.
fi

Definición 5.14 (BPP). *Un lenguaje $L \subseteq \Sigma^*$, con función característica $\chi_L : \Sigma^* \longrightarrow \{0,1\}$, se dice que pertenece a la clase* **BPP**[31] *si existe un par (p, N) donde:*

- *p es un polinomio univariado,*
- *N es una máquina de Turing determinística de tiempo polinomial,*

[31] *Bounded error probability polynomial time.*

de tal modo que para cada $x \in \{0,1\}^*$, la probabilidad de error del algoritmo probabilista diseñado en el modelo anterior verifica

$$\mathtt{Prob}_{y\in\{0,1\}^{p(|x|)}}[N(x,y) \neq \chi_L(x)] \leq 1/3,$$

asumiendo en $\{0,1\}^{p(|x|)}$ la distribución uniforme.

Esta clase asume la probabilidad de error a ambos lados, pero muchos de los algoritmos probabilistas no cometen errores en una de las respuestas (éste es el caso, por ejemplo, en los tests de primalidad probabilistas citados anteriormente). Éstas son las clases **RP** y co–**RP** siguientes:

Definición 5.15 (RP). *Con las anteriores notaciones, un lenguaje $L \subseteq \Sigma^*$ pertenece a la clase* **RP** *si existe un par (p, N) p es un polinomio univariado, y N es una máquina de Turing determinística de tiempo polinomial, de tal modo que para cada $x \in \{0,1\}^*$, se verifica:*

$$x \in L \Longrightarrow \mathtt{Prob}_{y\in\{0,1\}^{p(|x|)}}[N(x,y) = \text{accept}] \geq 2/3,$$

y

$$x \notin L \Longrightarrow \mathtt{Prob}_{y\in\{0,1\}^{p(|x|)}}[N(x,y) = \text{accept}] = 0.$$

La clase co–**RP** es la clase de lenguajes cuyos complementarios están en **RP**.

A modo de ejemplo, el Tests de Miller–Rabin es un algoritmo en **RP** pero no para el lenguaje de los número primos, sino para el lenguaje Comp $\subseteq \mathbb{N} := \{0,1\}^*$ formado por los números naturales que no son primos. El Test de Miller Rabin verifica que: si el input n es primo, entonces devuelve primo con probabilidad 1, mientras que si el input n es compuesto el algoritmo devuelve primo con probabilidad estrictamente menor que $1/2$. Es, de facto, como el test de Solovay y Strassen un "Test de Composición (Comp) en **RP**". Lo mismo sucede con los tests de [No]–Nulidad de polinomios citados anteriormente.

Estos modelos de algoritmo suelen recibir también el nombre de algoritmos de tipo **Monte Carlo**.

Pero, usualmente, se conocen algunas clases más finas de algoritmos llamados **Las Vegas** o **ZPP** (Zero error probability, expected polynomial time).

Definición 5.16. *Un lenguaje $L \subseteq \Sigma^*$ pertenece a la clase* **ZPP** *si existe un par (p, N) con las propiedades anteriores y se verifica:*

$$x \in L \Longrightarrow \mathtt{Prob}_{y\in\{0,1\}^{p(|x|)}}[N(x,y) = \text{accept}] = 1,$$

$$x \notin L \Longrightarrow \mathtt{Prob}_{y\in\{0,1\}^{p(|x|)}}[N(x,y) = \text{accept}] = 0,$$

y para cada $x \in \Sigma^*$, la esperanza de la función de tiempo es polinomial en la talla de x, esto es,

$$E_{y\in\{0,1\}^{p(|x|)}}[T_N(x,y)] \in |x|^{O(1)}.$$

Algunas primeras propiedades relacionando estas clases son las siguientes:

$$\mathbf{RP} \subseteq \mathbf{BPP},$$
$$\text{co–}\mathbf{RP} \subseteq \mathbf{BPP}.$$

En relación con el indeterminismo, se tienen las obvias relaciones:

$$\mathbf{RP} \subseteq \mathbf{NP},$$
$$\text{co–}\mathbf{RP} \subseteq \text{co–}\mathbf{NP}.$$

Es casi inmediato a partir de la definición que

Teorema 5.17.

$$\mathbf{ZPP} = \mathbf{RP} \bigcap \text{co–}\mathbf{RP}.$$

Demostración.– Supongamos que disponemos de una máquina M_1 que resuelve el L en **RP** y otra máquina M_2 que resuelve el mismo lenguaje en co–**RP**. Procedemos como sigue:

```
INPUT x ∈ Σ*
    while No hay respuesta do
        apply la máquina M1 sobre x,
        if M1 responde afirmativamente, OUTPUT: 1
        else do
            apply la máquina M2 sobre x,
            if M2 responde negativamente, OUTPUT: 0
            else return to while
            fi
        fi
    od
end
```

Es claro que esta combinación produce un algoritmo en **ZPP**. □

Pero, sin embargo, una pregunta difícil abierta es la siguiente

Problema Abierto 5.18. *Con las anteriores notaciones*

$$\mathbf{BPP} \subseteq \mathbf{NP}?$$

Lo que sí se conoce es:

Teorema 5.19. *Con las anteriores notaciones, se verifica:*

$$\mathbf{BPP} \subseteq \mathbf{PH},$$
$$\mathbf{BPP} \subseteq \mathbf{P}/\text{poly}.$$

5.5. Máquinas con Oráculos.

Definición 5.20. *Una máquina de Turing con oráculo $L \subseteq \Sigma^*$ es una máquina de Turing (determinística o no) que posee:*

- *una cinta especial en la que puede escribir llamada la cinta del oráculo,*
- *tres estados especiales $\{\text{QUERY}, q_{\text{yes}}, q_{\text{no}}\} \subseteq Q$*

cuyo funcionamiento es el siguiente:

En cualquier momento del cálculo, si la máquina accede al estado QUERY*, la máquina lee (en un sólo paso) el contenido $\omega \in \Sigma^*$ de la cinta del oráculo y devuelve o bien q_{yes} o q_{no} según $\omega \in L$ o no. Después sigue su computación.*

Para una clase de complejidad $\mathbf{C}$ y un lenguaje L, denotaremos por $\mathbf{C}^L$ la clase formada por todos los lenguajes acotados por el recurso descrito por $\mathbf{C}$, pero admitiendo máquinas con oráculo L. Así podemos definir las clases $\mathbf{P}^L, \mathbf{NP}^L$, etc...

Obviamente, si $L \in \mathbf{P}$, se tiene $\mathbf{P}^L = \mathbf{P}$ y así para cada clase de complejidad. La presencia de oráculos aumenta el poder computacional de una clase, aunque uno debe ser cuidadoso con su presencia.

Una de las primeras observaciones que se hicieron en torno al problema de Cook era la dificultad de utilizar argumentos basados en diagonalización (a la Gödel, Turing o como en los Teoremas de Jerarquía anterior): los argumentos basados en diagonalización tienen que ser "especiales" en la medida de que no son aplicables a máquinas de Turing con oráculos. Es el caso del siguiente resultado:

Teorema 5.21 ([Ba-Gi-So, 75])**.** *Existe un lenguaje A tal que*

$$\mathbf{P}^A = \mathbf{NP}^A = \mathbf{EXPTIME}^A.$$

Y también existe un lenguaje B tal que

$$\mathbf{P}^B \neq \mathbf{NP}^B.$$

Demostración.– Como resumen de la prueba, un lenguaje A que satisface el enunciado es el siguiente:

$$A := \{(M, x, n) \ : \ x \in L(M),\ T_M(x) \leq 2^n\}.$$

De otro lado, para un lenguaje B, definamos

$$U_B := \{1^n \ : \ \exists x \in B, |x| = n\}.$$

Claramente $U_B \in \mathbf{NP}^B$, pero se puede construir un lenguaje B tal que $U_B \notin \mathbf{P}^B$. El lenguaje se define del modo siguiente (diagonalización):

Para cada $i \in \{0,1\}^*$, sea M_i la máquina de Turing con oráculo B cuyo código es precisamente i. Definimos B inductivamente en función de i. En cada paso añadimos un número finito de elementos nuevos (o no añadimos ninguno).

Supongamos que ya hemos definido algunos elementos B_{i-1} de B en pasos anteriores. Ahora elijamos n mayor que la longitud de todos los elementos de B_{i-1} y ejecutamos la máquina M_i sobre 1^n. Consideramos todas las palabras que se guardan en la cinta del oráculo y alcanzan el estado QUERY. Si alguna está en B_{i-1} procedemos con q_{yes}, si alguna no ha sido predeterminada (no está en B_{i-1}), respondemos q_{no} y continuamos. Estamos ejecutando $M_i^{B_{i-1}}$, de hecho.

Detendremos la computación tras $2^n/10$ pasos.

Si la máquina $M_i^{B_{i-1}}$ termina su computación aceptando antes de realizar los $2^n/10$ pasos, escribiremos $B_i = B_{i-1}$ y, en particular, $1^n \notin U_B$. En caso contrario, elijamos una palabra $x \in \{0,1\}^*$, de longitud n, que no ha aparecido en la cinta del oráculo (existen porque el tiempo está acotado por $2^n/10$ y no hemos podido pasar por todos los $x \in \{0,1\}^n$) y definimos $B_i := B_{i-1} \cup \{x\}$.

Definimos finalmente $B := \cup_{i \in \mathbb{N}} B_i$. Con esta construcción garantizamos que la máquina M_i siempre devolverá una respuesta incorrecta sobre 1^n en menos de $2^n/10$ pasos. Por tanto, $U_B \notin \mathbf{P}^B$. □

6. La Frontera de lo Intratable: NP–Completitud

6.1. `Reducciones.` En la literatura se pueden encontrar varios conceptos de reducción. Una reducción es una simplificación de un problema en otro. Normalmente, en cuanto sigue, haremos referencia a reducciones de Karp (también llamadas polynomial-time many-one reductions); pero, por completitud mostraremos las tres reducciones descritas por los padres de la **NP**–completitud: Cook, Karp y Levin.

Definición 6.1 (Reducción de Cook)**.** *Dados dos lenguajes $L, L' \subseteq \Sigma^*$, decimos que L es Cook reducible a L' (también llamada reducción de Turing) si existe una máquina de Turing M con oráculo L', que finaliza sus computaciones en tiempo polinomial en el tamaño de la entrada, tal que el lenguaje aceptado por $M^{L'}$ es L.*

En esencia, se trata de lo siguiente: Para un input $x \in \Sigma^*$, el problema de pertenencia a L (i.e. evaluar $\chi_L(x)$) se resuelve mediante la aplicación de la máquina M con oráculo L' a x. Por ejemplo, si $L' \in \mathbf{P}$ y L es Cook reducible a L', entonces $L \in \mathbf{P}$.

Definición 6.2 (Reducción de Karp)**.** *Dados dos lenguajes $L, L' \subseteq \Sigma^*$, decimos que L es Karp reducible a L' si existe una función $f \in \mathbf{PF}$ (evaluable en tiempo polinomial) tal que para cada $x \in \Sigma^*$, se verifica:*

$$x \in L \iff f(x) \in L'.$$

De nuevo, para resolver el problema de pertenencia a L para un input $x \in \Sigma^*$, primero evaluamos $f(x)$ y luego aplicamos cualquier algoritmo que resuelva L' a $f(x)$. En particular, si $L' \in \mathbf{P}$ y si L es Karp reducible a L', entonces $L \in \mathbf{P}$.

Es claro que una reducción de Karp induce una reducción de Cook: La máquina con oráculo M es una máquina que evalúa f, que trata la cinta de output como cinta del oráculo y que, al alcanzar un estado final aceptador pasa al estado Query con oráculo L' y acepta si y solamente el oráculo devuelve aceptar.

Debe indicarse que no se sabe si las reducciones de Cook son más fuertes que las reducciones de Karp. De hecho, ni siquiera se sabe si la clase **NP** es estable por reducciones de Cook (aunque se sospecha que no es así).

Problema Abierto 6.3. *¿Es la clase* **NP** *cerrada bajo reducciones de Cook?*

6.1.1. Problemas de Búsqueda (Search Problem). Los problema de búsqueda y sus reducciones, fueron la motivación de la aproximación de Levin a la clase **NP**.

Un problema de búsqueda se define del modo siguiente: Sea $R \subseteq (\Sigma^*)^2$ una relación (en ocasiones variedad de incidencia o *solution variety*, según autores y contexto). Tenemos dos proyecciones canónicas $\pi_i^{(R)} : R \longrightarrow \Sigma^*$, $i = 1, 2$. Para cada $x \in \Sigma^*$ disponemos de dos fibras $(\pi_1^{(R)})^{-1}(x)$ y $(\pi_2^{(R)})^{-1}(x)$. Nos ocuparemos de la primera, aunque la segunda es simétrica para nuestras disquisiciones.

Por ejemplo, sea $\mathbb{F}_q$ un cuerpo finito y para cada lista de grados $(d) := (d_1, \ldots, d_n)$ consideremos $\mathcal{P}_{(d)}$ el conjunto formado por todas listas $(f_1, \ldots, f_n)$ de polinomios $f_i \in \mathbb{F}_q[X_1, \ldots, X_n]$ con $\deg(f_i) = d_i$, $1 \leq i \leq n$. Con una ordenación adecuada de coeficientes, tomando $\Sigma^* := \mathbb{F}_q$, podemos considerar la relación

$V_{(d)} \subseteq \mathcal{P}_{(d)} \times \mathbb{F}_q^n \subseteq (\Sigma^*)^2$ dada por la siguiente igualdad:

$$V_{(d)} := \{(f, x) \ : \ f(x) = 0\}.$$

La fibra $(\pi_1^{(R)})^{-1}(f)$ son las soluciones diofánticas (en $\mathbb{F}_q$) del sistema de ecuaciones definido por la lista f, mientras que $(\pi_2^{(R)})^{-1}(x)$ son los sistemas de ecuaciones que se anulan en $x \in \mathbb{F}_q$.

Dada una relación R, una función $\varphi : \Sigma^* \longrightarrow \Sigma^* \cup \{\emptyset\}$ resuelve el problema de búsqueda R si para cada $x \in \Sigma^*$ viene dada por:

$$\varphi(x) := \begin{cases} y \in (\pi_1^{(R)})^{-1}(x), & \text{para algún } y, \text{ si } (\pi_1^{(R)})^{-1}(x) \neq \emptyset \\ \emptyset & \text{en otro caso} \end{cases}$$

Es decir, φ devuelve algún punto de la fibra en el caso de fibra no vacía. En el caso anterior, una función que resuelve el problema de búsqueda definido por la primera proyección sería un resolvedor de ecuaciones polinomiales sobre cuerpos finitos, mientras que el problema de búsqueda simétrico sería un interpolador.

Un problema decisional asociado a un problema de búsqueda $R \subseteq \Sigma^*$, es el problema de decidir si la fibra es no vacía, es decir el lenguaje siguiente:

$$S_R := \{x \in \Sigma^* \ : \ \left(\pi_1^{(R)}\right)^{-1}(x) \neq \emptyset\}.$$

En un sentido amplio, tanto el Problema X de Hilbert como el Nullstellensatz de Hilbert son problemas decisionales asociados a problemas de búsqueda donde la relación es la variedad de incidencia de Room–Kempf o la *"solution variety"* de M. Shub y S. Smale, adaptada, en cada caso, al cuerpo correspondiente. Lo mismo puede decirse de problemas de optimización o de factibilidad de solución de sistemas de ecuaciones sobre los reales. En todo caso, Levin introdujo la siguiente reducción:

Definición 6.4 (Reducción a la Levin)**.** *Dados $R, R' \subseteq (\Sigma^*)^2$ dos problemas de búsqueda, una reducción de Levin de R a R' es un par de funciones (f, g) dadas mediante:*

- *La función $f : \Sigma^* \longrightarrow \Sigma^*$ es una reducción de Karp de S_R a $S_{R'}$, es decir, $f \in \mathbf{PF}$ y para cada $x \in \Sigma^*$, $x \in S_R$ si y solamente si $f(x) \in S_{R'}$.*
- *La función $g : D(g) \subseteq (\Sigma^*)^2 \longrightarrow \Sigma$, también está en $\mathbf{PF}$ y verifica que si $x \in S_R$ y si $x' = f(x)$ entonces,*

$$\forall y' \in (\pi_1^{(R')})^{-1}(x') \Longrightarrow (x, g(x, y')) \in R.$$

En suma, si R es Levin reducible a R' y si disponemos de un algoritmo polinomial que resuelve R', podemos resolver el problema de búsqueda para $x \in R$, comenzando con la aplicación de f, obteniendo $f(x)$. Si $f(x) \notin S_{R'}$, devolvemos $\emptyset$, en otro caso, resolvemos el problema de búsqueda para $f(x)$ (con respecto a R') obteniendo y' y terminamos devolviendo $g(x, y')$. Si R' se resuelve en tiempo polinomial, entonces R también se resuelve en tiempo polinomial.

No insistiremos mucho más en los problemas de búsqueda en este mini–curso. Indiquemos solamente que los problemas de búsqueda en la clase $\mathcal{PC}$ (relaciones

[con tallas] polinomialmente acotadas que admiten "checking" en tiempo polinomial) son Cook reducibles a problemas en **NP**. Véase [Go, 08] para un tratamiento más pormenorizado de los problemas de búsqueda en relación con la conjetura de Cook.

Dentro de la clase **P** y sus subclases (**LOG**, **NLOG**, **NC**,...) se suelen usar reducciones *log–space* (i.e. reducciones en **LOGF**,...). Sin embargo, esto se sale del contexto de este minicurso por lo que no insistiremos en esa dirección.

Definición 6.5. *Decimos que una clase de complejidad* **C** *es reducible (Cook, Karp, Levin, log–space...) a otra clase* **C**' *si los problemas de la primera son (Cook, Karp, Levin, log–space...) reducibles a problemas en la segunda.*

6.2. `El Teorema de Cook: Problemas` **NP**`-completos.` Aunque en ocasiones se usan reducciones de Cook para probar que ciertos problemas son **NP**–completos, nos restringiremos a las reducciones de Karp siempre que sea posible.

Definición 6.6. *Sea* **C** *una clase de complejidad, decimos que un lenguaje L es* **C***–duro para reducciones Karp (resp. Cook) si todos los lenguajes S de la clase* **C** *son Karp reducibles (resp. Cook reducibles) a L.*

Definición 6.7. *Sea* **C** *una clase de complejidad, decimos que un lenguaje L es* **C***–completo si verifica:*

- $L \in \mathbf{C}$
- *L es* **C***–duro.*

Proposición 6.8. *Sea $\mathbf{C}' \subseteq \mathbf{C}$ dos clases de complejidad y supongamos que* **C**' *es estable por reducciones à la Karp (resp. à la Cook). Sea L un lenguaje* **C***–completo, entonces*

$$L \in \mathbf{C}' \Longrightarrow \mathbf{C} = \mathbf{C}'.$$

Esta Proposición muestra la potencialidad de los problemas completos en una clase: ellos parecen condensar todo el potencial de la clase de complejidad y, por tanto, si "caen" a una clase menor, pero estable por las reducciones consideradas, toda la clase en la que son completas "cae" también en esa subclase. Diremos que ambas clases colapsan.

Pero, además, los problemas **NP**–completos adolecen del *don de la ubicuidad*. Se encuentran en casi cualquier ámbito de la computación. Lo que sigue es una galería de problemas **NP**–completos de diversos ámbitos del conocimiento. La galería no pretende ser completa, dado que se conocen miles de ejemplos, sino simplemente ilustrar algunos de esos casos. El lector interesado puede acudir al ya clásico [Ga-Jo, 79] o a la lisa/resumen de Wikipedia en

`http://en.wikipedia.org/wiki/List_of_NP-complete_problems`

Problema 6.9 (De la máquina universal a la **NP**–completitud)**.** *Consideremos el lenguaje K siguiente: Los elementos de K son listas (c_M, x, t), donde*

- *c_M es el código de una máquina de Turing indeterminística M (como en la máquina universal).*
- *$t \in \{1\}^*$ es una cota de tiempo en unario.*

- $x \in \Sigma^*$ *es una palabra tal que* $x \in L(M)$ *y* $T_M(L) \leq t$.

Teorema 6.10. *El lenguaje* K *anterior es* **NP***–completo.*

Demostración.– (Bosquejo) Para probar que K está en **NP** basta con aplicar la máquina Universal siguiendo t transiciones "plausibles" (adivinadas) a (c_M, x) y verificando que M acepta x en, a lo sumo, t pasos.

La reducción de cualquier lenguaje $L \in \mathbf{NP}$ a K consiste en escribir c_M (M es la máquina de Turing indeterminística que acepta L) y escribir $T_M(n)$ en unario. □

Recordemos el problema SAT descrito como Problema 5.8.

Teorema 6.11 ([Cook, 71])**.** *El problema* SAT *es* **NP***–completo.*

Demostración.– (Bosquejo) Es obvio que el problema está en la clase **NP**: basta con "adivinar" una asignación de verdad $x \in \{0,1\}^n$ y evaluar Φ en x. Aceptamos en el caso de que hayamos adivinado una asignación que satisface la fórmula.

Es más delicado probar que el problema es **NP**–completo. Basta con reducir K a SAT. La reducción consiste en escribir mediante fórmulas booleanas, el sistema de transición asociado a la máquina de código c_M y verificar el resto de las condiciones. □

Problema 6.12 (SAT-CNF)**.** *Recordemos el concepto de cláusula. Un literal* x *es una fórmula dada por una sola variable booleana* $x = X_i$ *o la negación de una variable booleana* $x = (\neg X_j)$*. Una cláusula es la disyunción de varios literales, es decir, una fórmula booleana*

$$\Phi(X_1, \ldots, X_n) = (x_1 \vee x_2 \vee \cdots \vee x_r)\,,$$

donde los x_i*'s son literales, es decir,* $x_i \in \{X_1, \ldots, X_n, (\neg X_1), \ldots, (\neg X_n)\}$*. Una fórmula booleana se dice en forma normal conjuntiva (*CNF*) si es la conjunción de variables cláusulas.*

El problema SAT-CNF *se define como el problema de las fórmulas* Φ *tales que* $\Phi \in \mathtt{SAT} \cap \mathtt{CNF}$*.*

Teorema 6.13 ([Cook, 71])**.** *El problema* SAT-CNF *es* **NP***–completo.*

Demostración.– (Bosquejo) Basta con observar que las fórmulas que salen de la reducción anterior se pueden escribir directamente en CNF. □

Un ejemplo particular son las fórmulas de Horn. Una cláusula de Horn es una cláusula en la que hay a lo sumo un literal positivo. Se trata de las cláusulas que pueden reescribirse como:

$$X_1, X_2, \ldots, X_n \Longrightarrow Y.$$

Una fórmula de Horn es una fórmula dada como la conjunción de varias cláusulas de Horn. Se tiene el sorprendente resultado siguiente (punto de partida de la Programación Lógica).

Teorema 6.14. *El lenguaje* HORN$\cap$ SAT *está en* **P***.*

Problema 6.15 (3SAT)**.** *Si bien no se puede reducir cualquier fórmula booleana a una fórmula de Horn, sí es factible reducir cualquier fórmula en* CNF *a una fórmula dada como la conjunción de cláusulas que involucran a lo sumo 3 literales en cada cláusula. Denotemos por* 3CNF *esas fórmulas y definamos el lenguaje* 3SAT *como las fórmulas en la intersección* SAT∩3CNF*.*

Teorema 6.16 ([Karp, 72])**.** *El problema* 3SAT *es* **NP***–completo.*

Demostración.– (Bosquejo) Basta con reproducir como cláusulas el proceso de evaluación de una fórmula SAT-CNF a 3SAT. □

Podemos también retomar el Nullstellensatz sobre cuerpos finitos. Así, podemos considerar $\mathcal{P}^K_{(3)}$ listas $f_1, \ldots, f_{n+1}$ de $n+1$ polinomios en n variables con grado acotado por 3 y coeficientes en el cuerpo K. Supongamos $K := \mathbb{F}_q$ es un cuerpo finito con q elementos y sea $\mathbb{K}$ la clausura algebraica de K.

Corolario 6.17. *El Hilbert Nullstellensatz* HN *sobre cuerpos primos es un problema* **NP***–duro y el siguiente es* **NP***–completo:*

$$\texttt{SAT} - \texttt{HN} := \{f = (f_1, \ldots, f_{n+1}) \in \mathcal{P}^K_{(3)} \ : \ \exists x \in K^n, f_1(x) = 0, \ldots, f_{n+1}(x) = 0\}.$$

Demostración.– En el Problema 2.3 se define el Nullstellensatz como el lenguaje:

$$\texttt{HN} := \{f = (f_1, \ldots, f_{n+1}) \in \mathcal{P}^K_{(3)} \ : \ \exists x \in \mathbb{K}^n, f_1(x) = 0, \ldots, f_{n+1}(x) = 0\}.$$

Se tratará de un problema **NP**–duro porque SAT es Karp reducible a él. Pero no se sabe si es **NP**–completo puesto que no podemos "adivinar" de manera controlada y simple las soluciones en $\mathbb{K}^n$ (recuérdese que $\mathbb{K}$ es un cuerpo de cardinal infinito) a partir de los coeficientes y, por tanto, no podemos garantizar que esté en **NP**.

En cambio sí está en **NP** su versión SAT-HN: la búsqueda de soluciones no ya en $\mathbb{K}^n$ sino en $K^n = \mathbb{F}_q^n$, que son fáciles de "adivinar". Como SAT también es reducible a SAT-HN, será un problema **NP**–completo. □

Problema 6.18 (CLIQUE)**.** *Un grafo $G := (V, E)$ se dice completo si E contiene todas las posibles aristas entre cualesquiera dos nodos de V. El lenguaje* CLIQUE *es el lenguaje dado por los pares (G, k) donde $G = (V, E)$ es un grafo y k es un entero positivo, de tal modo que G contiene un subgrafo completo de cardinal mayor o igual que k.*

Teorema 6.19 ([Karp, 72])**.** *El problema* CLIQUE *es* **NP***–completo.*

Demostración.– (Bosquejo) Reduciremos SAT-CNF a CLIQUE del modo siguiente. Supongamos que la fórmula Φ es la conjunción

$$\Phi := C_1 \wedge \cdots \wedge C_r,$$

donde $C^1, \ldots, C_r$ son cláusulas que involucran variables en $\{X_1, \ldots, X_n\}$ y literales $\{x_1, \ldots, x_k\}$. Definimos un grafo $G = (V, E)$ del modo siguiente:

- $V := \{(x_j, C_i) \ : \text{ el literal } x_j \text{ aparece en la cláusula } C_i\}$,
- $E := \{((x_j, C_i), (x_m, C_s) \ : \ C_i \neq C_s, \ \neg x_j \neq x_m\}$, donde hemos supuesto que la doble negación es la variable original.

- $k = r$ (i.e. el número de cláusulas).

Si el grafo G posee un subgrafo completo de cardinal mayor o igual que k, entonces la fórmula original es satisfactible. □

Problema 6.20 (`3-COLOR`)**.** *Se trata del lenguaje formado por los grafos no orientados $G := (V, E)$ que son tres coloreables, esto es, que se pueden asignar colores (etiquetas $\{1, 2, 3\}$) de tal modo que dos vértices vecinos no poseen el mismo color.*

Teorema 6.21 ([Karp, 72])**.** *El problema* `3-COLOR` *es* **NP***–completo.*

Demostración.– (Idea) Reducir `3SAT` a `3-COLOR`. □

Problema 6.22 (`HC` Hamiltonian Circuit)**.** *Un circuito hamiltoniano en un grafo orientado $G = (V, E)$ es una secuencia cerrada de vértices $\nu_1, \ldots, \nu_s$ de tal modo que se puede pasar de cada uno al siguiente $(\nu_i, \nu_{i+1}) \in V$ y $(\nu_s, \nu_1) \in V$. El lenguaje* `HC` *es el lenguaje formado por todos los grafos orientados G que poseen un circuito hamiltoniano pasando por todos los nodos del grafo.*

Teorema 6.23. *El problema* `HC` *es* **NP***–completo.*

Demostración.– (Idea) Reducir `3SAT` a `HC`. □

Problema 6.24 (`SUBSET SUM`)**.** *Se trata de las listas finitas de números enteros $\{a_i : 1 \leq i \leq n\} \subseteq \mathbb{Z}$ tales que existe $S \subseteq \{1, \ldots, n\}$ verificando*

$$\sum_{i \in S} a_i = 0.$$

Una variante de este problema es el Problema de la Mochila.

Problema 6.25 (`KANPSACK`)**.** *Dada una lista de enteros $\{a_i : 1 \leq i \leq n\} \subseteq \mathbb{Z}$ y dado $k \in \mathbb{N}$, decidir si*

$$\exists S \subseteq \{1, \ldots, n\}, \sum_{i \in S} a_i = k.$$

Observación 6.26. Ambos problemas pueden reescribirse como un problema de eliminación (Hilbert Nullstellensatz):

Decidir si el siguiente sistema de ecuaciones polinomiales con coeficientes en $\mathbb{Z}$ posee una solución en $\mathbb{C}^n$:

$$X_1^2 - X_1 = 0, X_2^2 - X_2 = 0, \ldots, X_n^2 - X_n = 0, k - \sum_{i=1}^{n} a_i X_i = 0.$$

Teorema 6.27. *Tanto* `SUBSET SUM` *como* `KNAPSACK` *son problemas* **NP***–completos.*

Problema 6.28 (`Minimum Distance`)**.** *En Teoría de Códigos Correctores de Errores trabajamos sobre un cuerpo finito $\mathbb{F}_q := GF_q$ que actúa como alfabeto. El espacio $\mathbb{F}_q^n$ es un espacio métrico con la distancia de Hamming:*

$$d_H(x, y) := \sharp\{i : 1 \leq i \leq n,\ x_i \neq y_i\}.$$

Un código es un subespacio lineal $C \subseteq \mathbb{F}_q^n$, que podemos definir mediante sus ecuaciones lineales por una matriz $H \in \mathcal{M}_{m\times n}(\mathbb{F}_q)$. La capacidad de un código viene determinada por el peso mínimo de las palabras en C o, equivalentemente, por el mínimo de las distancias de sus elementos.

Definimos el lenguaje `Minimum Distance` *como los pares (H, r) donde H es una matriz en $\mathcal{M}_{m\times n}(\mathbb{F}_q)$ y r es un número natural tal que existe x verificando*

$$Hx^t = 0, \;\; weight(x) := d_H(x, 0) \leq r.$$

Teorema 6.29 ([Va,97])**.** *El problema* `Minimum Distance` *es* **NP***–completo.*

Problema 6.30 (`ILP`)**.** *El problema de optimización lineal entera se define como el lenguaje de las cuaternas (M, a, c, g), donde $M \in \mathcal{M}_{m\times n}(\mathbb{Z})$ es una matriz con coordenadas enteras, $a \in \mathbb{Z}^m$ y $c \in \mathbb{Z}^n$ son vectores con coordenadas enteras y $g \in \mathbb{Z}$ es un entero positivo. Nos preguntamos si existe $x \in \mathbb{Z}^n$ tal que*

$$Mx^t \leq a, \; \langle c, x\rangle \geq g.$$

A pesar de que el problema de Programación lineal (con posibles soluciones racionales $x \in \mathbb{Q}^n$) se resuelve en tiempo polinomial (cf. [Kh, 79]) no es el mismo caso cuando se trata de soluciones diofánticas. El caso $0 - 1$ `ILP` es debido a [Karp, 72] y trata del caso particular en que las matrices tienen coordenadas en $\{0, 1\}$.

Teorema 6.31. *El problema* `ILP` *es* **NP***–completo.*

Por retomar la clase co–**NP** ya citada anteriormente, y el Problema `TAUT` descrito como Problema 5.12, tendremos

Teorema 6.32. *El Problema* `TAUT` *es co–***NP** *completo para reducciones de Karp.*

7. La Clase PSPACE

Si la esencia de la clase **NP** era la existencia de certificados cortos, la esencia de la clase **PSPACE** es la búsqueda de estrategias ganadoras en juegos de 2 personas con acceso completo a la información del juego. Si en el caso de **NP** es la presencia de cuantificadores existenciales, en la clase **PSPACE** será la alternancia entre cuantificadores existenciales y universales la clave sobre la estrategia (la estrategia ganadora para cualquier movimiento del oponente). Nos ocuparemos poco de esta clase en este curso (focalizado en la relación entre **P** y **NP**) y apenas si puntualizaremos unas pocas ideas esenciales, comenzando por los contenidos obvios, ninguno de los cuales se sabe si es o no estricto:

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PH} \subseteq \mathbf{PSPACE}.$$

Problema Abierto 7.1. *¿Es estricto alguno de los contenidos anteriores?*

7.1. `Problemas` **PSPACE-completos.** El ejemplo permanente de los problemas **PSPACE**–completos es `QBF`. Una fórmula booleana cuantificada es una fórmula de primer orden, en forma prenexa, involucrando cuantificadores existenciales y/o universales $\{\exists, \forall\}$, variables $\{X_1, \ldots, X_n, \ldots\}$ y conectivas booleanas $\{\neg, \vee, \wedge\}$. En suma, se trata de expresiones de la forma:

$$Q_1 X_1 Q_2 X_2 \cdots Q_n X_n \ \Phi(X_1, \ldots, X_n),$$

donde $Q_i \in \{\forall, \exists\}$ y Φ es una fórmula booleana libre de cuantificadores. Nótese que no posee variables libres (i.e. todas sus variables están cuantificadas).

Definición 7.2 (`QBF`)**.** *Se define el lenguaje* `QBF` *formado por todas las fórmulas booleanas que son tautología.*

Teorema 7.3. *Con las anteriores notaciones,* `QBF` *es* **PSPACE***–completo.*

Otros ejemplos de Problemas **PSPACE**–completos son:

- Problema de Palabra para gramáticas sensibles al contexto.
- Dada una expresión regular α, decidir si el lenguaje que describe $L(\alpha)$ coincide con Σ^*.
- Generalizaciones de juegos (extendidos a tableros $n \times n$) como Hex, Sokoban o Mah Jong,...

Si bien los dos primeros son relevantes en el procesamiento de lenguajes (dejando los lenguajes "tratables" en clases más simples como las libres de contexto), la gran variedad de juegos para los que se ha demostrado que la búsqueda de estrategias ganadoras es **PSPACE**–completo, da un cierto toque de popularidad y marketing del que pienso huir. El lector interesado bien puede acudir a Papadimitrou en [Pap, 94] o [Pap, 85] y continuadores.

Obviamente, una estrategia polinomial que resuelva cualquiera de esos problemas, implicaría la igualdad de todos los contenidos descritos al inicio de esta Sección. Pero es poco esperable.

7.2. `La Jerarquía Polinomial` **PH.** Dado un grafo $\mathcal{G} := (V, E)$, un *conjunto independiente* (también llamado estable) es un subconjunto de vértices $S \subseteq V$ de tal modo que dos vértices cualesquiera $x, y \in S$ no están unidos por ninguna arista (en E) del grafo $\mathcal{G}$. Es decir, el subgrafo inducido es totalmente aislado y tiene tantas componentes conexas como vértices.

El siguiente es un claro problema en la clase **NP**:

$$\texttt{IND} := \{(\mathcal{G}, k) \ : \ \text{existe un conjunto independiente } S \text{ con } \sharp(S) \geq k\}.$$

Pero podríamos convertirlo en un problema decisional del tipo siguiente:

$$\texttt{EXACT IND}\{(\mathcal{G}, k) \ : \ k \text{ es el máximo cardinal de subconjuntos independientes}\}.$$

El segundo no parece pertenecer a la clase **NP** sino que parece añadir un cuantificador universal $\forall$. La concatenación alternada de cuantificadores existenciales y universales da pie a la *Jerarquía Polinomial*: **PH**

Definición 7.4 (**PH**)**.** *Se definen las clases de lenguajes siguientes:*

- *Se dice que un lenguaje $L \subseteq \Sigma^*$ está en la clase Σ_i, $i \in \mathbb{N}$, $i \geq 1$, si existe un lenguaje $\mathcal{L}$ en* **P** *y existe un polinomio univariado q, de tal modo que una palabra $x \in \Sigma^*$, $|x| = n$, está en L si y solamente si*

  $$Q_1 u_1 \in \Sigma^{q(n)} Q_2 u_2 \in \Sigma^{q(n)} \cdots Q_i u_i \in \Sigma^{q(n)}, \ \ (x, u_1, \ldots, u_i) \in \mathcal{L},$$

  donde $Q_j \in \{\exists, \forall\}$, $Q_j \neq Q_{j+1}$, $Q_1 = \exists$.
- *Se dice que un lenguaje $L \subseteq \Sigma^*$ está en la clase Π_i, $i \in \mathbb{N}$, $i \geq 1$ si existe un lenguaje $\mathcal{L}$ en* **P** *y existe un polinomio univariado q, de tal modo que una palabra $x \in \Sigma^*$, $|x| = n$, está en L si y solamente si*

  $$Q_1 u_1 \in \Sigma^{q(n)} Q_2 u_2 \in \Sigma^{q(n)} \cdots Q_i u_i \in \Sigma^{q(n)}, \ \ (x, u_1, \ldots, u_i) \in \mathcal{L},$$

  donde $Q_j \in \{\exists, \forall\}$, $Q_j \neq Q_{j+1}$, $Q_1 = \forall$.

Como primeras observaciones se tiene:

$$\mathbf{NP} = \Sigma_1,$$
$$\text{co} - \mathbf{NP} = \Pi_1,$$
$$\Sigma_i \subseteq \Pi_{i+1},$$
$$\Pi_i \subseteq \Sigma_{i+1},$$
$$\Pi_i = \text{co} - \Sigma_i.$$

Definición 7.5 (PH). *Se denomina jerarquía polinomial* **PH** *a la clase dada mediante:*

$$\mathbf{PH} = \bigcup_{i=1}^{\infty} \Sigma_i.$$

Proposición 7.6 (Colapsos en **PH**)**.** *Se tienen los siguientes resultados:*

- *Si existiera i tal que $\Sigma_i = \Pi_i$, entonces* $\mathbf{PH} = \Sigma_i$ *(la jerarquía polinomial colapsaría al nivel i).*
- *Si* $\mathbf{P} = \mathbf{NP}$*, entonces* $\mathbf{PH} = \mathbf{P}$ *(la jerarquía polinomial colapsaría al nivel* **P***).*

No se sabe si la jerarquía polinomial posee problemas completos.

Problema Abierto 7.7. *¿Existe algún problema completo (para reducciones à la Karp) en la jerarquía polinomial?*

En cambio sí se conocen algunas de las consecuencias de su existencia

Proposición 7.8. *Si existiera algún problema completo para la clase* **PH***, entonces la jerarquía polinomial colapsaría a algún nivel i.*

En particular, es obvio que $\mathbf{PH} \subseteq \mathbf{PSPACE}$, pero si la jerarquía polinomial no colapsa, entonces $\mathbf{PH} \neq \mathbf{PSPACE}$. Porque ya hemos visto que sí hay problemas completos en **PSPACE**.

Existe una generalización de las máquinas indeterministas denominadas Máquinas de Turing que Alternan *ATM*. De la misma manera que el indeterminismo refleja la clase **NP**, las ATM modelizan las clase Σ_i y Π_i. Es conocido que el

tiempo polinomial en ATM's coincide con espacio polinomial; pero evitaremos esta discusión sobre ATM's y dirigiremos al lector a cualquiera de las referencias al uso.

7.3. `Circuitos:` $\mathbf{P}/\mathrm{poly}$. Un circuito booleano $\mathcal{C}$ es un grafo orientado y acíclico cuyos nodos están etiquetados de la manera siguiente:

- Los nodos de entrada están etiquetados con variables booleanas $X_1, \ldots, X_n$ y dos están específicamente etiquetados con las constantes booleanas $\{0, 1\}$.
- Los nodos interiores de fan–in 2, contienen una etiqueta de la forma (op, i_1, i_2), donde $op \in \{\vee, \wedge\}$, e i_1, i_2 son dos vértices "anteriores" (en una buena ordenación del grafo).
- Los nodos interiores de fan–in 1, contienen etiquetas de la forma $(\neg, i_1)$, e i_1 es un vértice "anterior".
- Hay un único nodo de output.

Los circuitos booleanos son programas finitos que evalúan funciones booleanas $f : \{0,1\}^n \longrightarrow \{0,1\}$. Si en lugar de haber usado las conectivas lógicas $\{\vee, \wedge, \neg\}$, hubiésemos identificado $\mathbb{F}_2 = \{0, 1\}$ y usado las operaciones de $\mathbb{F}_2$ como cuerpo, los hubiéramos llamado circuitos aritméticos o, incluso, *straight–line programs*. Esto nos llevaría a la *Teoría de la Complejidad Algebraica* y a otra interesantísima historia, que no es la pretendida en este curso. El lector interesado puede seguir las monografías [Bo-Mu, 72], [Bü-Cl-Sh, 97], [Bl-Cu-Sh-Sm, 98].

La talla del circuito es la talla del grafo y la altura (o profundidad) se suele identificar con la complejidad paralela (como en la clase **NC** de problemas bien paralelizables). Pero también éste es un asunto que se escapa de un curso como el presente.

Si $\mathcal{B}_n$ denota la clase de funciones booleanas con domino $\mathbb{F}_2^n$, C.E. Shannon y O. Lupanov ([Shn, 49], [Lu, 58]) demostraron que casi todas las funciones booleanas exigen circuitos de talla $2^n/n$ para ser evaluadas y que hay circuitos de esa talla para cualquier función booleana.

Recuérdese también que las funciones booleanas definen un $\mathbb{F}_2-$espacio vectorial y una $\mathbb{F}_2-$álgebra puesto que no son otra cosa que el anillo de clases de restos:

$$\beta_n := \mathbb{F}_2[X_1, \ldots, X_n]/\mathfrak{a},$$

donde $\mathfrak{a}$ es el ideal generado por $\{X_1^2 - X_1, \ldots, X_n^2 - X_n\}$. Un problema abierto relevante es el siguiente:

Problema Abierto 7.9. *Hallar una función booleana $\varphi \in \beta_n$ que necesite circuitos de talla $2^n/n$ para ser evaluada.*

Aquí haremos referencia solamente a la clase $\mathbf{P}/\mathrm{poly}$ que se define de la manera siguiente:

Definición 7.10 ($\mathbf{P}/\mathrm{poly}$)**.** *Se dice que un lenguaje $L \subseteq \{0,1\}^*$, está en la clase* $\mathbf{P}/\mathrm{poly}$ *si existe un polinomio univariado p y una familia de circuitos booleanos $\{\mathcal{C}_n : n \in \mathbb{N}\}$ de tal modo que:*

- *Para cada $n \in \mathbb{N}$, la talla del circuito $\mathcal{C}_n$ está acotada por $p(n)$.*

- *Para cada $n \in \mathbb{N}$, el circuito evalúa la función booleana dada como la función característica de $L_n \subseteq \{0,1\}^n$, donde*

$$L_n := \{x \in L \ : \ |x| = n\}.$$

Observación 7.11. Se tienen los siguientes contenidos:

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{P}/\text{poly}.$$

Teorema 7.12 (Karp–Lipton)**.** *Si* $\mathbf{NP} \subseteq \mathbf{P}/\text{poly}$*, entonces la jerarquía polinomial colapsa a* Σ_2*. Más aún, si* $\mathbf{EXPTIME} \subseteq \mathbf{P}/\text{poly}$*, entonces* $\mathbf{EXPTIME} = \Sigma_2$*.*

Más aún, si **P**=**NP**, entonces $\mathbf{EXPTIME} \not\subseteq \mathbf{P}/\text{poly}$.

8. Interactividad. IP=PSPACE

La interactividad es una generalización distinta del fenómeno de verificación de certificados que aparece intrínsicamente en la noción de indeterminismo. En lugar de trabajar con certificaciones, trataremos de trabajar con dos jugadores que intercambian información. Uno de los jugadores es el *Prover* (demostrador, identificable con el mago *Merlin* de la tradición artúrica). El prover M es un poderoso proveedor de pruebas/demostraciones que interactúa con el otro jugador. El otro jugador es el *verifier* (también identificable con Arturo, de la misma tradición literaria). El verifier A tiene capacidad computacional restringida y su actividad esencialmente consiste en tratar de verificar computacionalmente si la prueba aportada por Merlin es o no correcta. La interacción entre estos dos elementos es la que genera las clases **IP** y **AM**. En esencia el control de la capacidad computacional del prover es inexistente, mientras que son las restricciones impuestas al verifier lo que define la clase.

8.1. `Interactive Proof Systems.`

Definición 8.1. *Dadas dos funciones $f, g : \{0,1\}^* \longrightarrow \{0,1\}^*$, llamaremos interacción de k rondas entre f y g con input $x \in \{0,1\}^*$ a toda secuencia finita de palabras $a_0 = x, a_1, \ldots, a_k \in \{0,1\}^*$ dada mediante:*

$$\begin{array}{lcr} a_1 & := & f(x) \\ a_2 & := & g(x, a_1) \\ & \vdots & \\ a_{2i+1} & := & f(x, a_1, \ldots, a_{2i}) \\ a_{2(i+1)} & := & g(x, a_1, \ldots, a_{2i+1}) \\ & \vdots & \end{array}$$

donde hemos identificado cada $n-$tupla $(x, a_1, \ldots, a_i)$ con la palabra obtenida mediante adjunción $xa_1 \cdots a_i \in \{0,1\}^*$. A la $n-$tupla $(a_1, \ldots, a_k)$ se la llama transcripción de la interacción de k rondas.

El resultado (con respecto a f) de la interacción de $k = 2i+1$ rondas entre f y g sobre x se denota mediante $\text{out}_f\langle f, g\rangle(x) = a_{2i+1}$*. De modo análogo se denota el resultado (con respecto a g) de la interacción de $k = 2(i+1)$ rondas se denota mediante* $\text{out}_g\langle f, g\rangle(x) = a_{2(i+1)}$*.*

En los casos que nos interesan supondremos que las funciones f y g verifican que la talla de la imagen es polinomial en el tamaño del dato. Esto es, supondremos

$$|f(z)| \leq |z|^{O(1)}, |g(z)| \leq |z|^{O(1)}, \ \ \forall z \in \{0,1\}^*.$$

La nueva potencia de cálculo con interactividad se obtiene si aumentamos la capacidad computacional del verificador, admitiendo que sea una máquina de Turing probabilística. Esta es la noción siguiente.

Definición 8.2 (d**IP**). *Sea $k : \mathbb{N} \longrightarrow \mathbb{N}$ una función monótona creciente. Un lenguaje $L \subseteq \{0,1\}^*$ se dice que está en la clase $d\mathbf{IP}[k]$ (de lenguajes aceptados por un sistema determinístico de demostración interactiva, con número de rondas acotado por k) si existe una máquina de Turing determinista V que funciona en tiempo polinomial y tal que para cada input x el número de rondas está acotado por $k(|x|)$ y verifica:*

- **Completitud:** *Si $x \in L$, entonces existe $P : \{0,1\}^* \longrightarrow \{0,1\}^*$ tal que*
$$\mathrm{out}_V\langle V, P\rangle(x) = 1.$$
- **Validez:** *Si $x \notin L$, entonces para todo $P : \{0,1\}^* \longrightarrow \{0,1\}^*$*
$$\mathrm{out}_V\langle V, P\rangle(x) = 0.$$

A la función P se la denomina prover y a la máquina de Turing V se la denomina verifier.

Obsérvese que los valores con índice impar se puede suponer de un sólo dígito dado que el verificador es decisional (i.e. $a_{2i+1} \in \{0,1\}$).

Se define la clase

$$d\mathbf{IP} := \bigcup_{c \in \mathbb{N}} d\mathbf{IP}[n^c].$$

Teorema 8.3. *Se verifica la siguiente igualdad:*

$$d\mathbf{IP} = \mathbf{NP}.$$

Demostración.– Es claro que **NP** se identifica con los sistemas determinísticos interactivos con una sola ronda. Para el otro contenido basta con "recolectar" la transcripción de las interacciones (en número polinomial y de talla polinomial cada una) en un sólo "guessing" y recuperar la clase **NP**. □

El potencial de cálculo de los sistemas de demostración interactiva se observa mejor si reemplazamos la hipótesis determinística del verifier por una hipótesis probabilística.

Definición 8.4 (**IP**). *Sea $k : \mathbb{N} \longrightarrow \mathbb{N}$ una función monótona creciente. Un lenguaje $L \subseteq \{0,1\}^*$ se dice que está en la clase $\mathbf{IP}[k]$ (de lenguajes aceptados por un sistema de demostración interactiva, con número de rondas acotado por k) si existe un polinomio univariado p y una máquina de Turing probabilista V que funciona en tiempo polinomial y tal que para cada input x, $|x| = n$, genera aleatoriamente valores $r \in \{0,1\}^{p(n)}$ y con un número de rondas acotado por $k(|x|)$ y además verifica las dos propiedades siguientes:*

- **Completitud:** *Si* $x \in L$,$|x| = n$, *entonces existe* $P : \{0,1\}^* \longrightarrow \{0,1\}^*$ *tal que*
$$\texttt{Prob}_{\{0,1\}^{p(n)}}[\text{out}_V\langle V, P\rangle(x,r) = 1] \geq 2/3.$$
- **Validez:** *Si* $x \notin L$, *entonces para todo* $P : \{0,1\}^* \longrightarrow \{0,1\}^*$
$$\texttt{Prob}_{\{0,1\}^{p(n)}}[\text{out}_V\langle V, P\rangle(x,r) = 1] \leq 1/3.$$

La probabilidad considerada es la uniforme en ambos casos.

Se define la clase
$$\mathbf{IP} := \bigcup_{c\in\mathbb{N}} \mathbf{IP}[n^c].$$

Teorema 8.5 ([Shm, 92], [Lu-Fo-Ka, 92])**.** *Se verifica la siguiente igualdad:*
$$\mathbf{IP} = \mathbf{PSPACE}.$$

Observación 8.6. Obsérvese lo sutil de la diferencia entre **NP** y **PSPACE** en esta clasificación: la mera imposición del determinismo frente al probabilismo a la capacidad de trabajo del verifier.

Esto permite definir, de nuevo, el problema de la relación entre **NP** y **PSPACE** mediante

Problema Abierto 8.7. *Decidir si el siguiente contenido es estricto:*
$$d\mathbf{IP} \subseteq \mathbf{IP}.$$

8.2. `La prueba de la igualdad` **IP=PSPACE**. Nuestro objetivo aquí es demostrar el Teorema 8.5.

8.2.1. $\mathbf{IP} \subseteq \mathbf{PSPACE}$.
Demostración.– Sea $L \in \mathbf{IP}[n^c]$ un lenguaje en **IP**. Probaremos que $L \in \mathbf{PSPACE}$. La idea fundamental de esta prueba consiste en calcular para cada input $x \in \{0,1\}^*$, $|x| = n$, (mediante un algoritmo que sólo usa espacio polinomial) un "prover" que maximiza la probabilidad de aceptación para el input x dado. Se denomina la *estrategia del demostrador óptimo.*

Nótese que dado un input x y un candidato aleatoriamente elegido $r \in R := \{0,1\}^{p(n)}$, un prover P para (x,r) es simplemente una transcripción $(a_1, \ldots, a_{k(n)})$ tal que se verifica:
$$V(x, r, a_1, \ldots, a_{2i}) = a_{2i+1},$$
para cada i. En otras palabras el prover P para x y r viene dado por la secuencia con índice par: $(a_2, a_4, \ldots, a_{2i}, \ldots)$, mientras los impares son determinados por el verificador. De hecho, podemos considerar Γ el conjunto de las transcripciones $\gamma := (a_1, \ldots, a_t)$, con $t \leq T$, T y t impares, T impar maximal menor que n^c, y podemos definir recursivamente la función siguiente:
$$\varphi : R \times \Gamma \longrightarrow \{0,1\}^{n^{O(1)}}$$
mediante:
$$\varphi(x, r, \gamma) = m,$$
si y solamente si satisface las propiedades siguientes:

- La transcripción $\gamma = (a_1, \ldots, a_t)$, $t = 2k+1$, satisface
$$V(x, r, a_1, \ldots, a_{2i}) = a_{2i+1},$$
para todo i. Es decir, γ determina un prover compatible con x y r.
- Para $T \leq n^c$, T impar:
$$\mathrm{out}_V\langle V, \gamma m\beta\rangle(x, r) = V(x, r, \gamma, m, b_{2k+3}, \ldots, b_T) = 1,$$
donde
  - $\gamma m\beta := (a_1, \ldots, a_t, m, b_{2k+3}, \ldots, b_T)$,
  - $b_{2k+3} := V(x, r, \gamma, m)$,
  - $b_{2i+1} := V(x, r, \gamma, m, b_{2k+3}, \ldots, b_{2i})$,
  - $b_{2j} := \varphi(x, r, a_1, \ldots, a_t, m, b_{2k+3}, \ldots, b_{2j-1})$.

En esencia, φ actúa como un prover que conduce a aceptar a partir de x, r y una transcripción "parcial" γ.

Se dice que r es consistente con (x, γ, m) si se satisfacen las anteriores propiedades. Nótese que todas ellas se pueden expresar mediante una fórmula booleana cuantificada, con un número polinomial de cuantificadores (y de alternancias).

Finalmente, definimos la función
$$\Phi : \{0,1\}^* \times \Gamma \longrightarrow \{0,1\}^{|x|^{O(1)}},$$
dada mediante $\Phi(x, \gamma) := m$ si y solamente si m maximiza el número de $r \in R$, consistentes con (x, γ, m). Ahora se trata de probar que contar el número de candidatos aleatorios $r \in R$ consistentes con un cierto (x, γ, m) se puede hacer en **PSPACE**. Contar en **PSPACE** consiste en ir generando valores para las variables libres (que sólo aparecen en número polinomial) y verificar que una fórmula booleana cuantificada y prenexa sin variables libres es cierta o no (lo cual estaba en **PSPACE**) y acumular el número de casos positivos, antes de pasar al siguiente valor. Generando los diversos m's y γ's hallaremos el que maximiza la probabilidad de aceptar como aquél en el que hemos contado más casos positivos. Finalmente x es aceptado si la probabilidad máxima de aceptación es mayor que $2/3$ y rechazado si la probabilidad máxima de aceptación es menor que $1/3$. □

8.2.2. co–**NP**$\subseteq$ **IP**.

Demostración.– Probaremos que $\sharp$SAT están en **IP**. Recordemos que

$$\sharp\mathtt{SAT} := \{(\Phi, K) \ : \ \Phi \text{ dada en CNF y es satisfecha por } K \text{ instancias en } \{0,1\}^n\}.$$

La primera parte es la aritmetización de las fórmulas booleanas, especialmente en el caso de cláusulas. La idea es la identificación $\{0,1\} = \mathbb{F}_2$, donde $\mathbb{F}_2$ es el cuerpo de Galois de dos elementos. Las conectivas $\{\vee, \wedge, \neg\}$ se pueden transformar en polinomios en el cuerpo $\mathbb{F}_2[X, Y]$ de la forma obvia. Sin embargo, nos interesará trabajar con polinomios con coeficientes en $\mathbb{Z}$ y, más específicamente, con sus reducciones módulo un primo convenientemente elegido. Así que vamos a considerar p un primo suficientemente grande, $\mathbb{F}_p$ el cuerpo de Galois de orden p y una variedad algebraica cero–dimensional y $\mathbb{F}_p-$definible $V_n \subseteq \mathbb{F}_p^n$ dada mediante:
$$V_n := \{0,1\}^n = \{(x_1, \ldots, x_n) \in \mathbb{F}_q \ : \ x_i^2 - x_i = 0, 1 \leq i \leq n\},$$

Las conectivas, $\{\vee, \wedge, \neg\}$ definen respectivamente aplicaciones (y, por estar en cuerpos finitos, aplicaciones polinomiales) sobre V_n:

$$\wedge, \vee : V_2 \longrightarrow \mathbb{F}_p,$$
$$\neg : V_1 \longrightarrow \mathbb{F}_p.$$

Obviamente hay una infinidad de polinomios $p_\vee, p_\wedge \in \mathbb{F}_p[X.Y]$ y $p_\neg \in \mathbb{F}_p[X]$ que definen esas funciones polinomiales. Podemos hacer varias elecciones (usando el hecho de de $X_1^2 - X_1, X_2^2 - X_2, \ldots, X_n^2 - X_n$ es una base de Gröbner y hallando sus formas normales, por ejemplo). Por conveniencia usaremos las siguientes funciones:

$$p_\wedge(X, Y) := XY \in \mathbb{F}_p[X, Y],$$
$$p_\vee(X, Y) := 1 - (1 - X)(1 - Y) \in \mathbb{F}_p[X, Y],$$
$$p_\neg(X) := 1 - X.$$

La "ventaja" de esta elección es que, independientemente del cuerpo $\mathbb{F}_p$ (es decir, independientemente del primo), estos polinomios satisfacen que la imagen de V_2 (resp. V_1) está contenida en $\{0, 1\}$.

Dada una cláusula c de la forma:

$$c := \left(X_1^{\varepsilon_1} \vee X_2^{\varepsilon_2} \vee \cdots \vee X_n^{\varepsilon_n}\right),$$

donde $\varepsilon_i \in \{0, 1\}$ es de tal forma que:

$$X_i^{\varepsilon_i} := \begin{cases} X_i, & \text{si } \varepsilon_i = 1, \\ \neg X_i, & \text{en caso contrario.} \end{cases}$$

Entonces, definimos el polinomio asociado a la cláusula c como

$$p_c := 1 - \prod_{i=1}^{n} \varepsilon_i(X_i) \in \mathbb{F}_p[X_1, \ldots, X_n],$$

donde

$$\varepsilon_i(X_i) := \begin{cases} (1 - X_i), & \text{si } \varepsilon_i = 1, \\ X_i, & \text{en caso contrario.} \end{cases}$$

Nótese que $p_c(V_n) \subseteq \{0, 1\}$ y que $p_c(x_1, \ldots, x_n) = 1$ si y solamente si $c(x_1, \ldots, x_n) = 1$ (es decir si $(x_1, \ldots, x_n)$ es una instancia que satisface c).

Finalmente, dada $\Phi := \bigwedge_{i=1}^{s} c_i$ una fórmula del Cálculo Proposicional en forma normal conjuntiva, tenemos el polinomio

$$P_\Phi := \prod_{i=1}^{s} p_{c_i}(X_1, \ldots, X_n) \in \mathbb{F}_p[X_1, \ldots, X_n].$$

En el caso de tener cláusulas de tres variables cada una, tenemos un polinomio de grado 3 sobre la variedad V_n. Nótese que para una asignación $x \in V_n$, $P_\Phi(x) = 1$ si y solamente si x satisface Φ. Más aún, sigue siendo cierto que $P_\Phi(V_n) \subseteq \{0, 1\}$. Por tanto, para un primo p suficientemente grande, contar el número de asignaciones que satisfacen Φ, es lo mismo que calcular la traza del polinomio eliminante de P_Φ sobre V_n, esto es

$$Tr_{V_n}(P_\Phi) := \sum_{x \in V_n} P_\Phi(x).$$

Diseñaremos un sistema de demostración interactivo (con prover P y verificador V) que funciona del modo siguiente mediante $2n$ rondas:

- Para $i = 1$, denotemos por $p_1 \in \mathbb{F}_q[T]$ el polinomio univariado (de grado a lo sumo s) dado mediante:

$$p_1(T) := \sum_{(x_2,\ldots,x_n)\in V_{n-1}} P_\Phi(T, x_2, \ldots, x_n).$$

  Escribamos $v_1 := K$. El prover P aporta un polinomio $p_1' \in \mathbb{F}_q[T]$ de grado a lo sumo s. El verifier V, comprueba que $p_1'(0) + p_1'(1) = K$. En caso de respuesta negativa, rechaza y termina la computación (rechaza Φ). En caso de satisfacerse la igualdad, V genera aleatoriamente un valor $r_1 \in \mathbb{F}_q$, designa $v_2 := p_1'(r_1)$ y pasa a la siguiente interacción para decidir la igualdad:

$$\sum_{x\in V_{n-1}} P_\Phi(r_1, x) = p_1(r_1) = v_2.$$

- Para $i \geq 1$, supongamos que hemos definido la interacción mediante una secuencia

$$p_1', \ldots, p_{i-1}',$$

  y valores $r_1, \ldots, r_{i-1} \in \mathbb{F}_q$ de tal modo que
  - $p_j'(r_j) = p_j(r_j) = v_{j+1}$, $1 \leq j \leq i-1$,
  - $p_j'(0) + p_j'(1) = v_j$, $1 \leq j \leq i-1$.

  Denotemos por $p_i \in F_q[T]$ el polinomio dado mediante:

$$p_i := \sum_{x\in V_{i+1}} P_\Phi(r_1, \ldots, r_{i-1}, T, x).$$

  El prover P aporta un polinomio $p_i' \in \mathbb{F}_q[T]$ de grado a lo sumo s. El verifier V, comprueba que $p_i'(0) + p_i'(1) = v_i$. En caso de respuesta negativa, rechaza y termina la computación (rechaza Φ). En caso de satisfacerse la igualdad, V genera aleatoriamente un valor $r_i \in \mathbb{F}_q$, designa $v_{i+1} := p_i'(r_i)$ y pasa a la siguiente interacción.

Si $\Phi \in \sharp\mathtt{SAT}$, el prover genera una secuencia apropiada, dada mediante

$$p_i' := p_i, \quad 1 \leq i \leq n,$$

y ciertamente termina aceptando. Por tanto se tiene la **Completitud**.

En el caso $\Phi \notin \sharp\mathtt{SAT}$, si el proceso termina en aceptación es porque se ha generado una secuencia $p_1', \ldots, p_n'$ de polinomios de tal modo que $p_i' \neq p_i$ para todo i. Puesto que si $p_i' = p_i$ para algún i, también se tiene (volviendo hacia atrás) $p_1' = p_1$ y $p_1'(0) + p_1'(1) = v_1 = K$, con lo que Φ se satisfaría en K instancias $\Phi \in \sharp\mathtt{SAT}$. Pero además se ha seleccionado aleatoriamente una secuencia de valores $r_1, \ldots, r_n \in \mathbb{F}_q$ de tal modo que $p_i'(r_i) = p_i(r_i)$ para todo i. En particular, tenemos que la probabilidad de que el protocolo responda afirmativamente para $\Phi \notin \sharp\mathtt{SAT}$ está acotada por la probabilidad de que la secuencia de polinomios $p_i' - p_i \neq 0$

tenga un cero en $\mathbb{F}_q$. Es fácil usar argumentos al uso para acotar esa probabilidad por

$$1 - \left(1 - \frac{s}{q}\right)^n.$$

Eligiendo q suficientemente grande (en relación con s y n) tenemos la condición de **Solidez**. □

8.2.3. **PSPACE $\subseteq$ IP**. Se trata de probar que $\texttt{QBF} \in \mathbf{IP}$.

Demostración.– La prueba es análoga a la del caso de $\sharp\texttt{SAT}$. La diferencia aquí es que tenemos una fórmula booleana cuantificada, en forma prenexa y sin variables libres. Esto es,

$$\Phi := Q_1 X_1 Q_2 X_2 \cdots Q_n X_n,\ \varphi(X_1, \ldots, X_n), \tag{2}$$

donde $Q_i \in \{\forall, \exists\}$ y φ es una fórmula del Cálculo Proposicional que podemos suponer en forma normal conjuntiva de cláusulas con tres variables cada una.

Ahora nos interesa evaluar la siguiente cantidad (módulo un primo suficientemente grande):

$$P_\Phi := (\Lambda_1)_{x_1 \in \{0,1\}} (\Lambda_2)_{x_2 \in \{0,1\}} \ldots (\Lambda_n)_{x_n \in \{0,1\}} P_\varphi(x_1, \ldots, x_n), \tag{3}$$

donde

$$(\Lambda_i)_{x_i \{0,1\}} := \begin{cases} \sum_{x_i \in \{0,1\}}, & \text{si } Q_i = \exists, \\ \prod_{x_i \in \{0,1\}}, & \text{si } Q_i = \forall. \end{cases}$$

Nótese además que P_Φ es exactamente la cantidad de puntos $x \in V_n$ que satisfacen la fórmula Φ. Por tanto, la condición en $\mathbb{F}_p$ a verificar es $P_\Phi \neq 0$ ($P_\Phi > 0$, si estuviéramos en $\mathbb{Z}$).

Hay dos problemas aparentes. De una parte, viendo P_Φ como polinomio el grado es excesivamente grande (cada vez que aparece un cuantificador universal $\forall$ introducimos un producto y, por tanto, duplicamos el grado). Esto difiere notablemente del caso de $\sharp\texttt{SAT}$ anterior: allí sólo había cuantificadores existenciales, luego sólo aparecía $\sum$. De otro lado, podría ser que el número P_Φ como número entero fuera extraordinariamente grande (del orden de 2^{2^n}). Esta segunda dificultad no es tal. Echando un vistazo a un clásico como [Ib-Mo, 83] para observar que la no nulidad de números dados por esquemas de evaluación y valor absoluto 2^{2^n} se puede testar probabilísticamente con unos pocos primos p de valor absoluto acotado por 2^{2n}. La misma idea sirve en este caso.

Ahora procedemos de modo análogo al caso de $\sharp\texttt{SAT}$ con la única diferencia de que los cuantificadores existenciales $\exists x_i$ nos llevan a un sumatorio $\sum_{x_i \in \{0,1\}}$ y el verificador tiene que testar $p'_i(0) + p'_i(1) = v_{i-1}$, mientras que en el caso de cuantificadores universales $\forall x_i$, aparece un producto $\prod_{x_i \in \{0,1\}}$ y el verificador tendrá que testar $p'_i(0) p'_i(1) = v_{i-1}$. Sin embargo, la dificultad que aparece es que no tenemos control sobre el grado de p'_i que debe proveer el prover. Si el grado siguiera las pautas de la fórmula (3) anterior, parecería que el grado del polinomio $p'_i(X_i)$ tiene que ser 2^n, lo que haría imposible trabajar al verificador en tiempo polinomial. Aquí hay dos maneras plausibles de atacar la dificultad. De

una parte, dado que $x_i^2 - x_i$ para cada i, la forma normal de P_Φ módulo la base de Gröbner $X_1^2 - X_1, \ldots, X_n^2 - X_n$ tiene grado a lo sumo 1 en cada variable, con lo que podríamos conformarnos con que el prover nos ofrezca polinomios univariados p_i' de grado a lo sumo 2.

Otra alternativa (la original de [Shm, 92]) consiste en transformar nuestra fórmula original (2) de tal modo que pierda el carácter canónico de su forma, pero nos dé una nueva fórmula que no estará en forma prenexa sino que cuantificadores y variables se entremezclan, pero tienen la propiedad de que cada variable X_i está a la izquierda y derecha de, a lo sumo, un cuantificador universal. De ese modo, podemos garantizar que el grado en la variable X_i del polinomio P_Φ es, a lo sumo $2\deg(P_\varphi)$ y nos conformamos con que p_i' sea de grado a lo sumo $2\deg(P_\varphi)$.

La transformación es del tipo siguiente:

- Trabajaremos de izquierda a derecha sobre la fórmula inicial

$$\Phi := Q_1 X_1 Q_2 X_2 \cdots Q_n X_n,\ \varphi(X_1, \ldots, X_n),$$

  Mientras $Q_i = \exists$, no hacemos nada, hasta encontrar el caso con $Q_i X_i = \forall X_i$.
- Supongamos que tenemos:

$$\Phi := \exists X_1 \exists X_2 \cdots \exists X_{i-1} \forall X_i \tau(X_1, \ldots, X_n),$$

  siendo

$$\tau(X_1, \ldots, X_n) := Q_{i+1} X_{i+1} \cdots Q_n X_n,\ \varphi(X_1, \ldots, X_n).$$

  Entonces, transformamos la fórmula Φ en una nueva fórmula en la que habremos introducido nuevas variables $Y_1, \ldots, Y_n$ y viene dada por:

$$\Phi := \exists X_1 \cdots \forall X_i \exists Y_1 \cdots \exists Y_i \left[\bigwedge_{k=1}^{i} (X_i = Y_i) \right] \wedge \tau(Y_1, \ldots, Y_i, X_{i+1}, \ldots, X_n).$$

Repitiendo el proceso (de izquierda a derecha) con el siguiente cuantificador universal, habremos introducido a lo sumo $O(n^2)$ variables nuevas y la talla de la nueva fórmula es del orden del cuadrado de la talla de la fórmula Φ dada originalmente.

La propiedad de que ninguna variable esté a la izquierda y derecha de más de un cuantificador universal nos permite garantizar que el polinomio univariado $p_i(X_i)$ correspondiente al caso de una variable X_i, afectada de un cuantificador universal $\forall$, tiene grado a lo sumo $2\deg(P_\varphi)$, está acotado linealmente en la talla de Φ y podremos proceder de la manera indicada. □

9. La Demostración de Dinur del Teorema **PCP** (bosquejo)

Por simplicidad admitiremos el alfabeto binario en cuanto sigue $\Sigma = \{0, 1\}$. Identificaremos un número natural $i \in \mathbb{N}$ con su codificación binaria $i \in \Sigma^*$.

En la definición de la clase **IP** ya encontramos una primera discusión sobre verificadores. Aquí vamos a insistir en el concepto de manera más detallada.

Un verificador es, en realidad, una máquina de Turing con oráculo, en la que la cinta del oráculo pasa a llamarse cinta de direcciones (*address tape*), que admite solamente oráculos finitos identificados con palabras $\pi \in \{0,1\}^*$ que se denominarán, de acuerdo con las discusiones sobre Interactividad, demostraciones (*proofs*). Se usa el término *acceso inmediato a la proof* para designar el proceso siguiente: cada vez que en la cinta de direcciones aparezca el número natural i, la máquina puede acceder a la coordenada $i-$ésima $\pi[i]$ de la demostración π.[32] Por ejemplo, una máquina de Turing indeterminista que caracteriza un lenguaje $L \in \mathbf{NP}$, es una máquina determinística con oráculo finito que se permite acceder un número polinomial de veces a las coordenadas de la demostración (o certificado) y que, en tiempo polinomial, decide si el input, y la porción de prueba a la que hemos tenido acceso, son aceptados o no.

En esta Sección admitiremos verificadores probabilistas definidos del modo siguiente.

Definición 9.1 ((r,q)–Verifier)**.** *Sea $L \subseteq \{0,1\}^*$ un lenguaje y sean $q, r : \mathbb{N} \longrightarrow \mathbb{N}$ dos funciones. Decimos que L tiene un verificador (r,q) si existe una máquina de Turing probabilista V (verificador) que funciona en tiempo polinomial y satisface las siguientes propiedades:*

- *Para una entrada $x \in \{0,1\}^*$ de longitud n, V genera aleatoriamente una palabra $\rho \in \{0,1\}^*$ de longitud $|\rho| \leq r(n)$ (el "guessing").*
- *V posee una cinta especial donde están contenidas palabras $\pi \in \{0,1\}^*$ que denominaremos pruebas.*
- *V posee otra cinta especial que denominaremos Access Tape. En esa cinta V puede escribir un dígito $i \in \{0,1\}^*$, interpretarlo cono número natural (i.e. $i \in \mathbb{N}$) y acceder al lugar $i-$ésimo de la prueba mediante un estado especial llamado* Query*. De esto modo, una vez escrito i, accede en tiempo constante al dígito $\pi[i]$ de la prueba π.*
- *El número de veces en que se utiliza el estado* Query *está acotado por $q(n)$.*

El resultado del cálculo de V con guessing ρ y prueba π se denotará mediante $V^\pi(x,\rho)$. Adicionalmente, se han de verificar las siguientes dos propiedades para $x \in \{0,1\}^*$:

- **Completitud:** *Si $x \in L$, entonces $\exists \pi \in \{0,1\}^*$ tal que:*

$$\texttt{Prob}_{\rho \in \{0,1\}^{r(n)}}[V^\pi(x,\rho) = 1] = 1.$$

- **Solidez:** *Si $x \notin L$, entonces $\forall \pi \in \{0,1\}^*$ se verifica:*

$$\texttt{Prob}_{\rho \in \{0,1\}^{r(n)}}[V^\pi(x,\rho) = 1] \leq 1/2.$$

Definición 9.2. *Decimos que un lenguaje L está en la clase $\mathbf{PCP}[r,q]$ si posee un verificador (r_1,q_1) con $r_1 \in O(r)$ y $q_1 \in O(q)$.*

[32] En esencia es el mismo proceso que en el caso del oráculo: aquí π es el grafo de la función característica de un lenguaje finito $\Pi \subseteq \{x : |x| \leq k\}$, para algún k. Así, $\pi[i] = 1$ si, y sólo si, $i \in \Pi$.

Teorema 9.3 (**Teorema PCP**). *Se verifica la siguiente igualdad:*

$$\mathbf{NP} := \mathbf{PCP}[\log(n), 1].$$

El primer resultado en la interacción entre verificadores y clases indeterministas es la prueba del contenido $\mathbf{NEXP} \subseteq \mathbf{PCP}[poly(n), poly(n)]$, y fue demostrado en [Ba-Fo-Lu, 90]. A partir de este resultado se inicia un período de intensa actividad que culmina en los trabajos [Ar-Lu-Mo-Su-Sz, 98] y [Ar-Sa, 98]. Todos ellos recibieron el Premio Gödel de 2001 como consecuencia de estos trabajos. La historia de este Teorema puede seguirse, por ejemplo, en

`http://www.cs.princeton.edu/~dmoshkov/courses/pcp/pcp-history.pdf`

Aquí nos ocuparemos de dar un resumen de la prueba más reciente obtenida por I. Dinur en [Dinur, 07].[33]

Antes de avanzar en la prueba, discutamos algunos resultados parciales.

Proposición 9.4. *Se verifica:*

$$\mathbf{PCP}[r, q] \subseteq \mathbf{NTIME}(2^{O(r)}q).$$

Demostración.– Basta con hacer:

INPUT: $x \in \{0,1\}^*$, $|x| = n$
guess $\pi \in \{0,1\}^{q(n)}$
for each $\rho \in \{0,1\}^{r(n)}$
eval $V^\pi(x, \rho)$
end

El tiempo del verificador V es polinomial en $r(n)$ y n. Lo ejecutamos $2^{r(n)}q(n)$ veces, luego el tiempo está acotado por $2^{O(r)}q$. □

Así concluimos la primera inclusión (el contenido débil) del Teorema **PCP**:

Corolario 9.5. *Se verifica:*

$$\mathbf{PCP}[\log(n), 1] \subseteq \mathbf{NP}.$$

La parte dura será probar el otro contenido.

9.1. PCP `Algoritmos Aproximativos y Brechas.`

Definición 9.6 (Val)**.** *Sea φ una fórmula booleana en forma normal conjuntiva, dada por la conjunción de m cláusulas. Definimos* $\mathrm{val}(\varphi)$ *del modo siguiente: Sea $R(\varphi)$ el máximo de los cardinales de los subconjuntos S del conjunto de las cláusulas de φ tales que $\wedge_{i\in S}\varphi_i$ es satisfactible (es decir, el máximo número de cláusulas en φ que definen una fórmula satisfactible). Definimos:*

$$\mathrm{val}(\varphi) := \frac{R(\varphi)}{m}.$$

[33]Debe señalarse que las fechas de publicación de los trabajos tienden a ser muy posteriores a las fechas de "obtención" o "anuncio" de los resultados (FoCS, SToCS, etc.). Por ejemplo, se suele decir que el Teorema **PCP** es de 1992, mientras que la prueba de Dinur es de 2005. Usaremos siempre la fecha de publicación del resultado definitivo con sus pruebas completas.

Obviamente, una fórmula φ es satisfactible si y solamente si $\text{val}(\varphi) = 1$.

Definición 9.7 (Algoritmo Aproximativo para `MAX 3SAT`)**.** *Sea $\rho \in \mathbb{R}$, $0 < \rho \leq 1$. Un algoritmo A se dice que es $\rho-$Aproximativo para φ si toma como entrada una fórmula φ en forma normal conjuntiva con 3 variables por cláusula (3CNF) y devuelve una instancia $x \in \{0,1\}^n$ tal que el número de cláusulas de φ que son satisfactibles en x es, al menos, $\rho\,\text{val}(\varphi)m$.*

Ejemplo 9.8 (Un algoritmo $1/2-$aproximativo para `MAX 3SAT`)**.** *Se trata de un algoritmo greedy obvio. Trabajamos variable tras variable. Supongamos que las variables de φ son $X_1, \ldots, X_n$. Ahora, comenzando con $i = 1$ y siguiendo hasta $i = n$ hacemos la siguiente selección voraz:*
Tomemos $S := \{\varphi_1, \ldots, \varphi_s\}$ las cláusulas que nos quedan por tratar. Definamos S_0 como el subconjunto de las cláusulas en S que se satisfacen con $X_i = 0$ y $S_1 := S \setminus S_0$. Obviamente, las cláusulas en $S_1 b$ son las cláusulas que se satisfacen tomando $X_i = 1$. Elijamos la $i-$ésima coordenada de la instancia x como:

$$x[i] := \begin{cases} 0, & \text{si } \sharp(S_0) \geq \sharp(S_1), \\ 1, & \text{en otro caso.} \end{cases}$$

Pasemos al caso $i + 1$ con

$$S := \begin{cases} S_0, & \text{si } x[i] = 1, \\ S_1, & \text{en otro caso.} \end{cases}$$

Obviamente, con esta asignación, en cada iteración vamos guardando más cláusulas que las que restan y la conjunción de todas las que vamos guardando es satisfactible en la instancia que vamos construyendo. Por eso podemos asegurar que es un algoritmo $1/2-$aproximativo para `MAX 3SAT`*.*

Una de las consecuencias del Teorema **PCP** para los algoritmos aproximativos es el siguiente:

Corolario 9.9. *Existe una constante $\rho < 1$ tal que si existiera un algoritmo $\rho-$aproximativo para* `MAX 3SAT`*, entonces $\mathbf{P} = \mathbf{NP}$.*

Es decir, no podemos encontrar milagros con algoritmos de optimización aproximativos para problemas como los propuestos. Una alternativa interpretativa son los *gap problems*, problemas de brecha, como el siguiente:

Definición 9.10 (`GAP 3SAT`)**.** *Sea $\rho \in \mathbb{R}$, $0 \leq \rho < 1$. El problema $\rho-$*`GAP 3SAT` *es el problema de determinar, para una fórmula φ en forma normal conjuntiva con cláusulas de 3 variables, lo siguiente:*

- *Si φ es satisfactible, entonces φ está en el lenguaje $\rho-$*`GAP 3SAT`*.*
- *Si $\text{val}(\varphi) < \rho$, entonces φ no está en en lenguaje $\rho-$*`GAP 3SAT`*.*

Se dice que un algoritmo A resuelve el $\rho-$`GAP 3SAT` *si responde afirmativamente para fórmulas satisfactibles y responde negativamente para fórmulas tales que $\text{val}(\varphi) < \rho$.*

Nótese que, deliberadamente, no hemos exigido a nuestro algoritmo que responda correctamente en el caso de fórmulas que no son válidas y satisfacen $\text{val}(\varphi) \geq \rho$.

Definición 9.11. *Sea $\rho \in \mathbb{R}$, $0 \leq \rho < 1$. Decimos que $\rho-$*GAP 3SAT *es* **NP***–duro si para cada lenguaje $L \in$ **NP**, existe una función $f \in$ **PF** tal que:*

- *Si $x \in L$, entonces $f(x)$ es una entrada con respuesta positiva para el $\rho-$*GAP 3SAT*,*
- *si $x \notin L$, entonces $f(x)$ es una entrada con respuesta negativa para el $\rho-$*GAP 3SAT*.*

9.2. CSP: Problemas de Satisfacción de Restricciones.

Definición 9.12 (q$\mathtt{CSP}_W$)**.** *Se define el lenguaje q$\mathtt{CSP}_W$ como el conjunto de listas finitas $\varphi := (f_1, \ldots, f_m)$, donde $f_i : (\mathbb{Z}/W\mathbb{Z})^q \longrightarrow \{0,1\}$ son funciones (llamadas restricciones), donde $\mathbb{Z}/W\mathbb{Z}$ es el anillo de restos módulo W en el anillo $\mathbb{Z}$. Una instancia $u \in (\mathbb{Z}/W\mathbb{Z})^q$ se dice que satisface una restricción f_i si $f_i(u) = 1$.*
Se define la validez $\mathrm{val}\,(\varphi)$ *de manera análoga a como hicimos para el caso de* MAX SAT *y decimos que una lista $\varphi \in q\mathtt{CSP}_W$ es satisfactible si* $\mathrm{val}\,(\varphi) = 1$.

Ejemplo 9.13 (Ecuaciones Polinomiales)**.** *Dado que para cada i, $f_i^{-1}(0)$ y $f_i^{-1}(1)$ son dos conjuntos algebraicos (finitos). Uno podría interpretar f_i como la función característica de un subconjunto $V_i \subseteq (\mathbb{Z}/N\mathbb{Z})^q$ dado por ecuaciones polinomiales. En el caso $q = 3$, $W = 2$ tenemos obviamente que el conjunto de las fórmulas en forma normal conjuntiva dadas por cláusulas de 3 variables son ejemplos de listas finitas en $3\mathtt{CSP}_2$.*

Normalmente omitiremos el sub-índice $_W$ en el caso $W = 2$ y escribiremos

$$\mathrm{q}\mathtt{CSP} = \mathrm{q}\mathtt{CSP}_2.$$

También podemos definir los problemas de GAP como en el caso de fórmulas booleanas como se hizo en la subsección anterior, y disponer de $\rho -$ GAP qCSP como problema. En ese caso, el Teorema **PCP** es equivalente a la siguiente formulación:

Teorema 9.14 (Teorema **PCP**)**.** *Existe un número real ρ, $0 < \rho < 1$, tal que $\rho-$*GAP *q*CSP *es* **NP***–duro.*

Proposición 9.15. *Las formulaciones en los Teoremas 9.3 y 9.14 son equivalentes.*

Demostración.– Se prueba:

- **Teorema 9.3 $\Longrightarrow$ Teorema 9.14:** La idea es que si **NP**$\subseteq$ **PCP**$[\log(n), 1]$, entonces $1/2-$GAP qCSP es **NP**–duro. Para ello basta con reducir cualquier problema **NP**–completo a $1/2-$GAP qCSP y, de hecho, basta con reducir 3SAT a $1/2-$GAP qCSP.
- **Teorema 9.14 $\Longrightarrow$ Teorema 9.3:** Si $\rho-$GAP qCSP fuera **NP**–completo, para algún ρ y para alguna constante q, bastaría con traducirlo a sistema **PCP** con q *queries*, solidez acotada por ρ y aleatoriedad logarítmica para cualquier lenguaje L (via las reducciones). El verificador V esperará una prueba π como una posible asignación a las variables de las restricciones f_i ($W = 2$). Entonces, el verificador elige aleatoriamente un $i \in \{1, \ldots, m\}$, donde m es el número de restricciones, que se escribe con un número

logarítmico de dígitos. Realiza q *queries* y trata de verificar si f_i es satisfactible en la instancia obtenida de π. Si $x \in L$ el verificador siempre acertará con probabilidad 1. En caso contrario, aceptará con probabilidad a lo sumo ρ. Para subir la probabilidad de la solidez hasta un medio, basta con realizar la estrategia un número constante k de veces de manera independiente para obtener $(1-\rho)^k \geq 1/2$.

□

El Corolario 9.9 es consecuencia de la siguiente consecuencia del Teorema 9.14.

Corolario 9.16. *Existe una constante $\rho \in \mathbb{R}$, $0 < \rho < 1$ tal que $\rho-$*`GAP 3SAT` *es* **NP***–duro.*

Omitiremos la prueba.

9.3. `Reducciones CL.` Comenzaremos introduciendo un nuevo tipo de reducciones.

Definición 9.17 (Reducciones CL)**.** *Una aplicación F de listas de* `CSP` *en listas de* `CSP` *se denomina reducción CL (viene de complete linear–blowup) si $F \in$* **PF** *y verifica las siguientes propiedades:*

- **Completitud:** *Si φ es una lista de restricciones satisfactible, entonces $F(\varphi)$ también es satisfactible.*
- **Solidez:** *Si φ es una lista de restricciones de aridad q, sobre un alfabeto de talla W ($\mathbb{Z}/W\mathbb{Z}$) con n variables y m restricciones, la nueva lista $\psi := F(\varphi)$ verifica:*
  - *ψ tiene $C(q,W).m$ restricciones,*
  - *la talla del alfabeto de ψ es dada por $\mathcal{W}(q,W)$,*

  donde $C(q,W)$ y $\mathcal{W}(q,W)$ son funciones que dependen de q y W, pero son independientes de n y m.

El resultado fundamental de la prueba de [Dinur, 07] del Teorema **PCP** es el siguiente lema:

Lema 9.18 (Main Lemma)**.** *Existen constantes $q_0 \geq 3$ y $\varepsilon_0 > 0$ y una reducción CL F verificando:*
Para cada lista φ de restricciones en q_0`CSP` *(i.e. restricciones de aridad q_0 sobre alfabeto binario) la lista $\psi := F(\varphi)$ está también en q_0*`CSP` *(es decir, misma aridad y mismo alfabeto) y verifica además que para cada ε, $0 < \varepsilon < \varepsilon_0$ se tiene:*

$$\mathrm{val}\,(\varphi) \leq 1-\varepsilon \Longrightarrow \mathrm{val}\,(\psi) \leq 1-2\varepsilon.$$

Este lema implica el Teorema **PCP**, como se muestra en el párrafo siguiente:

Demostración del Teorema 9.14 a partir del Lema 9.18. Vamos a probar que $(1-2\varepsilon_0)-$`GAP` q_0`CSP` es **NP**–duro, donde q_0, ε_0 y F son los del Lema 9.18. Para empezar, como `3SAT` está contenido entre las entradas de q_0`CSP` (porque $q_0 \geq 3$), sabemos que q_0`CSP` es **NP**–duro. Por tanto, bastará con hallar una reducción en **PF** de q_0`CSP` a $(1-2\varepsilon_0)-$`GAP` q_0`CSP`. Para ello, sea φ una lista de m restricciones de aridad q_0 y alfabeto binario y sea $k := \lfloor \log_2 m \rfloor + 1$. Apliquemos, entonces

$\psi := F^k(\varphi)$, es decir, k iteraciones de la reducción CL F. Obsérvese que si φ es satisfactible también lo es ψ (por ser F una reducción CL). De otro lado, si φ no es satisfactible, entonces $\operatorname{val}(\varphi) \leq 1 - 1/m$. Por tanto, tras k iteraciones de F, usando el lema, tendremos

$$\operatorname{val}(\psi) \leq 1 - min\{2\varepsilon_0, 2^k/m\} = 1 - 2\varepsilon_0.$$

Finalmente, como q_0 y el tamaño del alfabeto permanecen constantes, también es constante $C := C(q_0, 2)$. Por ello, la talla de ψ está acotada por $C^k m = m^{\log_2 C+2}$ y es polinomial en la talla de φ. Por último, las k iteraciones de F se evalúan en tiempo polinomial puesto que la talla de los resultados intermedios es polinomial y $F \in \mathbf{PF}$. Hemos concluido que φ es una lista de restricciones en $(1 - 2\varepsilon_0)-\mathtt{GAP}$ $q_0\mathtt{CSP}$ y éste problema es **NP**–duro. □

9.4. `La Prueba del Lema 9.18 (bosquejo).` La prueba del Lema 9.18 reposa, a su vez, en dos lemas técnicos conocidos como *Gap Amplification* y *Alphabet Reduction*:

Lema 9.19 (Gap Amplification)**.** *Para todo $\ell \in \mathbb{N}$, $\ell > 1$, existe una CL–reducción*

$$G_\ell : q\mathtt{CSP} = q\mathtt{CSP}_2 \longrightarrow 2\mathtt{CSP}_W,$$

que transforma listas φ de restricciones de aridad q sobre el alfabeto binario en listas de restricciones de aridad 2 sobre un alfabeto $\mathbb{Z}/W\mathbb{Z}$ de tal modo que existe un número real positivo $\varepsilon_0(\ell, q)$ (dependiente solamente de ℓ y q) tal que para todo $\varepsilon < \varepsilon_0(\ell, q)$ y toda lista φ en $q\mathtt{CSP}$:

$$\operatorname{val}(\varphi) \leq 1 - \varepsilon \Longrightarrow \operatorname{val} G_\ell(\varphi)) \leq 1 - \ell\varepsilon.$$

Lema 9.20 (Alphabet Reduction)**.** *Existe una constante positiva $q_0 \in \mathbb{N}$ y una CL-reducción*

$$H : 2\mathtt{CSP}_W \longrightarrow q_0\mathtt{CSP},$$

tal que para cada lista en $2\mathtt{CSP}_W$ se verifica:

$$\operatorname{val}(\varphi) \leq 1 - \varepsilon \Longrightarrow \operatorname{val} H(\varphi)) \leq 1 - \frac{\varepsilon}{3}.$$

Para obtener el Lema 9.18 se combinan los dos lemas anteriores usando $\ell = 6$ y considerando la $CL-$reducción $f := h \circ g_\ell$. Por tanto, la prueba se reduce a la prueba de estos dos lemas aún más técnicos.

La contribución más original de la propuesta de I. Dinur es el Lema 9.19. La idea consiste fundamentalmente en reducirse al caso de restricciones que involucran a lo sumo 2 variables cada una, aún al precio de aumentar considerablemente el tamaño del alfabeto. Esto, en principio es sencillo. Si nuestra lista de restricciones iniciales es $\varphi := (\varphi_1, \ldots, \varphi_m)$ y considero una de ellas $\varphi_i(X_1, \ldots, X_q)$, dependiendo de q variables, en realidad tengo una aplicación $\varphi_1 : \{0,1\}^q \longrightarrow \{0,1\}$. Puedo perfectamente reinterpretar la caja $\{0,1\}^q = \mathbb{Z}/2\mathbb{Z}^q$ como $\mathbb{Z}/2^q\mathbb{Z}$ (aún al precio de perder alguna estructura de grupo subyacente), tengo un nuevo alfabeto de tamaño exponencial y una nueva variable Y_i que representa a los elementos de ese alfabeto. Ahora recuerdo que φ_i es una función característica y la reemplazo por varias funciones características nuevas $\{\psi_{i,j}\}$ definidas del modo siguiente. De una

parte, "recuerdo" que tengo una identificación de $\mathbb{Z}/2^q\mathbb{Z}$ con $\mathbb{Z}/2^q$ y me permito considerar "proyecciones" $\pi_j : \mathbb{Z}/2^q\mathbb{Z} \longrightarrow \mathbb{Z}/2\mathbb{Z}$ que asocian a cada $y \in \mathbb{Z}/2^q\mathbb{Z}$ su coordenada $j-$ésima $\pi_j(y) \in \mathbb{Z}/2\mathbb{Z}$. Finalmente, puedo identificar $\{0,1\}$ como subconjunto de $\mathbb{Z}/q\mathbb{Z}$ y definir para cada $(y,z) \in \mathbb{Z}_q\mathbb{Z}^2$

$$\psi_{i,j}(y,z) = 1 \Longleftrightarrow \left\{ \begin{array}{lcl} \phi_i(y) & = & 1 \\ \pi_j(y) & = & z \\ z & \in & \{0,1\} \end{array} \right.$$

Esta sencilla reducción mantiene la satisfactibilidad, es computable en **PF** y permite un cierto control de la validez, sin aumentar excesivamente el número de restricciones. Sin embargo, ha aumentado considerablemente nuestro alfabeto, pero ha reducido la aridad. Esto significa que, posteriormente, deberemos disminuir la aridad y ese es el rol del Lema 9.20. Pero disponer de restricciones con sólo dos variables nos permite trabajar con una estructura interna de gran importancia:

Definición 9.21 (Grafo de Restricciones)**.** *Sea $\varphi := (\varphi_1, \ldots, \varphi_m)$ una lista de restricciones en $2\mathtt{CSP}_W$ que depende de variables $X_1, \ldots, X_n$, aunque cada restricción de la lista sea de aridad 2. Definimos un grafo (de hecho, multi–grafo porque admitiremos multiplicidades en las aristas) $G := (V, E)$ asociado, al que llamaremos Grafo de Restricciones de φ en los términos siguientes:*

- *Los vértices (o nodos) del grafo $V := \{1, \ldots, n\}$ están asociados a las variables.*
- *Para cada restricción φ_i, si depende de las variables X_{i_1}, X_{i_2}, entonces sumaremos 1 a la multiplicidad de la arista $(i_1, i_2) \in E$ de nuestro grafo.*
- *En el caso en el que φ_i sólo dependa de una variable X_{i_1} sumaremos 1 a la multiplicidad de la arista $(i_1, \ldots, i_1) \in V$.*

Es relativamente fácil obtener una estructura de grafo $q-$regular asociado a una lista de restricciones e, incluso, una estructura de *Expander*. Esto lleva nuestro análisis a la teoría de los expanders.

Los expanders merecen un estudio tan detallado como la conjetura de Cook y supondrían un curso adicional tan denso como el aquí propuesto. Unas orientaciones generales sobre lecturas en torno a los "expanders" pueden ser las siguientes (también sus referencias)'.

La prueba de Dinur es simplemente un argumento más en defensa de los grafos regulares con alta conectividad. Entre otros pueden destacarse, las aplicaciones a construcción de redes informáticas o telefónicas de gran conectividad, en el diseño de algoritmos, en códigos correctores de errores, en generadores pseudo-aleatorios, en análisis de derrandomización, etc. Es destacable su uso en la prueba de la igualdad **SLOG**= **LOG** del trabajo [Re, 08].

Entre algunas referencias generalistas podemos destacar [Lu-Ph-Sa, 88], el excelente trabajo [Re-Va-Wi, 02], la nota [Sar, 04] y, sobre todo, el excelente *survey* [Ho-Li-Wi, 06]. Las notas de un curso de Widgersion sobre expanders pueden consultarse en

`http://www.math.ias.edu/~ boaz/ExpanderCourse/`

La prueba de I. Dinur del lema sobre "Gap Amplification", diseña una reducción CL novedosa, que se denomina Potenciación ("Powering"). Dicha reducción transforma listas de reducciones φ en $\mathtt{2CSP}_W$, cuyo grafo es un expander, en una nueva lista φ^t en $\mathtt{2CSP}_{W'}$, sobre un nuevo alfabeto W'. La nueva lista contiene una 2–restricción por cada camino de longitud $t+\sqrt{t}$ en el grafo de restricciones original y sus dos variables se interpretarán como funciones definidas en bolas de radio $t+\sqrt{t}$ dentro del grafo. Esta reducción aumenta el alfabeto y el número de restricciones ("amplification"), aunque de un modo "controlable" en el sentido de las reducciones CL. A su vez, esta reducción permite dominar la validez de la nueva lista de restricciones, gracias al especial comportamiento de la distribución de probabilidad de los caminos aleatorios ("random walks") en el expander de restricciones original. En la exposición del curso trataremos de hacer algunas precisiones más sobre este proceso.

Referencias

[Ah-Ho-Ul, 75] A.V. Aho, J.E. Hopcroft, J.D. Ullman. *"The Design and Analysis of Computer Algorithms".* Addison–Wesley, 1975.

[Ah-Ul, 95] A.V. Aho, J.D. Ullman. *Foundations of Computer Science.* W. H. Freeman, 1995.

[Ar-Ba, 09] S. Arora, B. Barak. *"Computational Complexity: A Modern Approach".* Cambridge University Press, 2009.

[Ar-Lu-Mo-Su-Sz, 98] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, *Proof verification and the hardness of approximation problems*, Journal of the ACM **45** (1998), 501–555.

[Ar-Sa, 98] S. Arora, S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, Journal of the ACM **45** (1998), 70–122.

[Ba-Fo-Lu, 90] L. Babai, L. Fortnow, C. Lund, *Nondeterministic exponential time has two-prover interactive protocols.* In Proc. of the 31st Annual Symp. Found. of Comput. Sci. (FoCS), IEEE Comput. Soc., 1990, 16–25.

[Ba-Gi-So, 75] T. Baker, J. Gill, R. Solovay. *Relativizations of the* $\mathbf{P} =? \mathbf{NP}$ *question.* SIAM J. on Comput. **4** (1975) 431–442.

[Ba-Dí-Ga, 88] J.L. Balcazar, J.L. Díaz and J. Gabarró. *"Structural Complexity I"*, EATCS Mon. on Theor. Comp. Sci. **11**, Springer, 1988.

[Ba-Dí-Ga, 90] J.L. Balcázar, J.L. Díaz and J. Gabarró. *"Structural Complexity II"*, EATCS Mon. on Theor. Comp. Sci., Springer, 1990.

[Be-Prd, 06] C. Beltrán, Luis M. Pardo. *On the Complexity of Non Universal Polynomial Equation Solving: Old and New Results.* Foundations of Computational Mathematics: Santander 2005. L. Pardo, A. Pinkus, E. Sülli, M. Todd (Eds.), Cambridge University Press, 2006, 1–35.

[Be-Prd, 09a] C. Beltrán, Luis M. Pardo, *Smale's 17th problem: average polynomial time to compute affine and projective solutions*, J. Amer. Math. Soc. **22** (2009), 363–385.

[Be-Prd, 09b] C. Beltrán, Luis M. Pardo, *Efficient polynomial system solving by numerical methods*, J. of Fixed Point Theor. & App. **6** (2009), 65-85.

[Be-Prd, 11] C. Beltrán, Luis M. Pardo, *Fast linear homotopy to find approximate zeros of polynomial systems*, Found. of Comput. Math. **11** (2011) 95–129.

[Bl-Cu-Sh-Sm, 98] L. Blum, F. Cucker, M. Shub, and S. Smale, *"Complexity and real computation"*, Springer-Verlag, New York, 1998.

[Bl, 67] M. Blum. *A machine independent theory of the complexity of recursive functions.* Journal of the ACM **14**, N.2, (1967) 322–336.

[Bo-Cr-Wi, 09] C. Bonatti, S. Crovisier, A. Wilkinson. *The C1-generic diffeomorphism has trivial centralizer.* Publications mathématiques de l'IHÉS **109** (2009) 185–244.

[Bo, 72] A. Borodin. *Computational complexity and the existence of complexity gaps.* J. of the ACM **19** (1972) 158–174.

[Bo-Mu, 72] A. Borodin, J. Munro. *"The Computational Complexity of Algebraic and Numeric Problems"*. Elsevier, NY, 1972.

[Bü-Cl-Sh, 97] P. Bürguisser, M. Clausen, M. Amin Shokrollahi. *"Algebraic Complexity Theory"*. Grundlehren del mathematischen Wissenschaften, vol. **315**, Springer, 1997.

[Co, 65] A. Cobham. *The intrinsic computational difficulty of functions*. In Proc. Logic, Methodology, and Philosophy of Science II (Proc. 1964 Internat. Congr.), North Holland (1965) 24–30.

[Cook, 71] S. Cook. *The complexity of theorem–proving procedures*. In Proc. 3rd Ann. ACM Symp. on Theor. Comput Sci. (1971) 151–158.

[Da-He, 88] J. Davenport, J. Heintz. *Real quantifier elimination is doubly exponential,* J. Symbolic Computation **5** (1988) 29–35.

[Dinur, 07] I. Dinur. *The PCP theorem by gap amplification*. J. of the ACM **54**, vol 3 (2007), Art. 12.

[Ed, 65a] J. Edmonds. *Minimum partition of a matroid into independent sets*, J. of Res. of the Nat. Bureau of Standards (B) **69** (1965) 67–72.

[Ed, 65b] J. Edmonds. *Maximum mathcing and a polyhedron with 0,1–vertices*, J. of Res. of the Nat. Bureau of Standards (B) **69** (1965) 125–130.

[Ed, 65b] J. Edmonds. *Paths, tress and flowers*. Canad. J. of Math. **17** (1965) 449–467.

[Fo-Ho, 03] L. Fortnow, S. Homer. *A Short History of Computational Complexity*. Bull. of the Europ. Ass. for Theor. Comput. Sci. **80**, June 2003.

[Lu-Fo-Ka, 92] L. Fortnow, C. Lund, H. Karloff. *Algebraic methods for interactive proof systems*. J. of the ACM **39** (1992) 859–868. Anunciado, con N. Nisan de co-autor adicional, en Proceedings of 31st Symposium on the Foundations of Computer Science. IEEE, New York, 1990, pp. 290.

[Ga-Jo, 79] M.R. Garey, D.S. Johnson. *"Computers and Intractability: A Guide to the Theory of NP–Completness"*. W.H. Freeman, 1979.

[Gi-He-Prd *et al.*, 97] M. Giusti, K. Hägele, J. Heintz, J.E. Morais, J.L. Montaña, L.M. Pardo. *Lower bounds for diophantine approximations*. J. of Pure and App. Algebra **117 & 118** (1997) 277–317.

[Go, 99] O. Goldreich. *"Modern Cryptography, Probabilistic Proofs and Pseudo–Randomness"*. Algorithmc and Combinatorics **17**, Springer, 1999.

[Go, 08] O. Goldreich. *"Computational Complexity: A Conceptual Approach"*. Cambridge University Press, 2008.

[Ha-Mo-Prd-So, 00] K. Hägele, J.E. Morais, L.M. Pardo, M. Sombra. *The intrinsic complexity of the Arithmetic Nullstellensatz*. J. of Pure and App. Algebra **146** (2000) 103–183.

[Ha-St, 65] J. Hartmanis, R. Stearns. *On the computational complexity of algorithms*. Trans. of the A.M.S. **117** (1965) 285–306.

[Ha-Le-St, 65] J. Hartmanis, P. M. Lewis II, R. E. Stearns. *Hierarchies of memory limited computations*. In Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logical Design, 1965, 179–190.

[He-Sc, 82] J. Heintz and C.P. Schnorr. *Testing polynomials which are easy to compute*. In "Logic and Algorithmic (an International Symposium in honour of Ernst Specker)", Monographie n. **30** de l'Enseignement Mathématique (1982) 237–254. A preliminary version appeared in *Proc. 12th Ann. ACM Symp. on Computing* (1980) 262–268.

[He-St, 66] F. Hennie and R. Stearns. *Two–tape simulation of multitape Turing machines*. J. of the ACM, **13** (1966) 533–546.

[Ho-Li-Wi, 06] S. Hoory, N. Linial, A. Widgerson. *Expander graphs and their applications*. Bulletin (New series) of the Amer. Math. Soc. **43** (2006) 439–561.

[Ib-Mo, 83] O.H. Ibarra, S. Moran. *Equivalence of Straight–Line Programs*. J. of the ACM **30** (1983) 217–228.

[Karp, 72] R. Karp. *Reducibility among combinatorial problems*. In "Complexity of Computer Computations" (R.E. Miller & J.W. Hatcher, eds.), Plenum Press, 1972, 85–103.

[Kh, 79] L. Khachiyan. *A polynomial algorithm in linear programming*. Soviet Math. Doklady **20** (1979) 191-194.

[Kn, 68–05] D.E. Knuth. *"The Art of Computer Programming"*, vols. 1 to 4.4. Recently re–edited by Addison–Wesley, 1997–2005.

[Ko, 92] D.C. Kozen. *"The Design and Analysis of Algorithms"*. Texts and Monographs in Computer Science, Springer Verlag, 1992.

[Kr-Prd, 96] T. Krick, L.M. Pardo. *A computational method for diophantine approximation*. En Algorithms in Algebraic Geometry and Applications, Proc. MEGA'94, Progress in Mathematics **143**, Birkhäuser Verlag, 1996, 193–254.

[Kr-Prd-So, 01] T. Krick, L.M. Pardo, M. Sombra. *Sharp estimates for the Arithmetic Nullstellensatz*. Duke Math. J. **109** (2001) 521–598.

[Le, 87] H.W. Lenstra. *Factoring integers with elliptic curves*. Annals of Math. **126** (1987) 649–673.

[Le-Le-Ma-Po, 90] A.K. Lenstra, H.W. Lenstra, M.S. Manasse, J.M. Pollard. *The number field sieve*. En Proc. 22nd ACM Symp. on Theory of Comput. (1990) 564–572.

[Lev, 73] L.A. Levin. *Universal search problems*. Probl. Pred. **7** (1973) 115–116. (Traducido al inglés en Proble. Inf. Trans. **9** (1973) 265–266).

[Lu-Ph-Sa, 88] A. Lubotzky, R. Phillips, and P. Sarnak. *Ramanujan graphs*. Combinatorica, **8** (1988) 261–277.

[Lu, 58] O. B. Lupanov. *A method of circuit synthesis*. Izvesitya VUZ, Radiofizika **1** (1958) 120–140.

[Ma-Me, 82] E. Mayr and A. Meyer. *The complexity of the word problem for commutative semigroups*. Advances in Math. **46** (1982) 305–329.

[Mi, 76] G.L. Miller. *Riemann's hypothesis and tests for primality*. J. Comput. Syst. Sci. **13** (1976) 300–317.

[Mo-Ra, 95] R. Motwani, P. Raghavan. *"Randomized Algorithms"*. Cambridge University Press, 1995.

[Pap, 85] C.H. Papadimitrou. *Games against nature*. J. Comput. Syst. Sci. **31** (1985) 288–301.

[Pap, 94] C. H. Papadimitrou. *"Computational Complexity"*. Addison–Wesley, 1994.

[Prd, 95] L.M. Pardo. *How lower and upper complexity bounds meet in elimination theory*. In Proc. AAECC–11, Paris 1995, G. Cohen, M.Giusti and T. Mora, eds., Springer LNCS **948**, 1995 33–69.

[Ra, 60] M. O. Rabin. *Degree of difficulty of computing a function and a partial ordering of recursive sets*. Tech. Rep. No. **2**, Hebrew University, 1960.

[Ra, 66] M. O. Rabin. *Mathematical theory of automata*. In Proc. of 19th ACM Symposium in Applied Mathematics, 1966, 153–175.

[Ra, 80] M.O. Rabin. *Probabilistic algorithms for testing primality*. J. Number Theory **12** (1980) 128–138.

[Ra-Su, 07] J. Radhakrishnan, M. Sudan, *On Dinur's proof of the PCP Theorem*. Bull. of the AMS **44** (2007) 19–61.

[Re, 08] O, Reingold. *Undirected connectivity in log-space.* Journal of the ACM **55** (2008), 1–24.

[Re-Va-Wi, 02] O. Reingold, S. Vadhan, A. Wigderson. *Entropy waves, the zig-zag graph product, and new constant-degree expanders.* Ann. of Math. **155** (2002) 157–187.

[Ri-Sh-Ad, 78] R.L. Rivest, A. Shamir y L.A. Adleman. *A method for obtaining digital signatures and public-key cryptosystems*. Comm. ACM **21** (1978) 120–126.

[Sar, 04] P. Sarnak. *What is an Expander?* Notices of the Amer. Math. Soc. **51** (2004) 762–763.

[Sa, 70] W.J. Savitch. *Relationships between nondeterministic and deterministic tape complexities*. J. Comput. System. Sci. **4** (1970) 177–192.

[Sc-Ve, 94] A. Schönhage,E. Vetter. *"Fast Algorithms. A Multitape Turing machine Implementation"*. Wissenschaftverlag, 1994.

[Sc, 80] J.T. Schwartz. *Fast probabilistic algorithms for verification of polynomial identities*. J. of the ACM **27**, (1980), 701–717.

[Shm, 92] A. Shamir. *IP = PSPACE*. J. of the ACM **39** (1992) 869–877.

[Shn, 49] C.E. Shannon. *The synthesis of two-terminal switching circuits*. Bell System Technical J. **28** (1949) 59–98.

[Smale, 00] S. Smale, *Mathematical problems for the next century*, Mathematics: frontiers and perspectives, Amer. Math. Soc., Providence, RI, 2000, pp. 271–294.

[So-St, 77] R. Solovay, V. Strassen. *A fast Monte Carlo test for primality.* SIAM J. on Comput. **6** (1977) 84–85.

[Tr, 64] B. Trakhtenbrot. *Turing computations with logarithmic delay* (in Russian). Algebra and Logic **3** 33–48.

[Tu, 02] W. Tucker (2002). *A Rigorous ODE Solver and Smale's 14th Problem.* Found. of Comput. Math. **2** (2002) 53–117.

[Va,97] A. Vardy. *Algorithmic Complexity in Coding Theory and the Minimum Distance Problem.* In Proc. STOC'97 (1997) 92–109.

[Wa-We, 86] K. Wagner, G. Wechsung. *"Computational Complexity"* . D. Reidel (1986).

[Zi, 90] R. Zippel. *Interpolating polynomials from their values.* J. Symbol. Comput. **9** (1990) 375–403.

Departamento de Matemáticas, Estadística y Computación, Facultad de Ciencias, Universidad de Cantabria, Avda. Los Castros s/n, 39005 Santander

Correo electrónico: luis.m.pardo@gmail.com