Universidad Complutense de Madrid

Trabajo de fin de grado en Matemáticas

Doble Grado en Ingeniería Informática y Matemáticas

Búsqueda de funciones booleanas complejas: construcciones mediante monotonía y relación entre repetitividad y endogamia

Autora:
Celia Rubio Madrigal

Directores:
Ismael Rodríguez Laguna
Narciso Martí Oliet

Curso 2021/2022
Madrid, a 30 de junio de 2022

Resumen

El problema de la identificación de funciones booleanas que requieren circuitos grandes para ser computadas ha tenido pocos avances desde su concepción hace ya 70 años. En este trabajo proponemos nuevos conceptos y técnicas que podrían ayudar en su búsqueda. Entre ellas, presentamos la endogamia como una propiedad que los circuitos polinómicos podrán poseer en mayor medida. Buscaremos su correlación experimental con nociones de repetitividad de las funciones computadas, que habremos definido previamente a partir de diagramas de decisión. Por último, mostraremos una técnica para encontrar cotas inferiores del tamaño de circuitos que hará uso del concepto de monotonía. Con esta técnica demostraremos una cota de $2n$ puertas de anchura en el primer nivel de un tipo de circuitos concreto, que creemos puede aumentarse al considerar el resto del circuito.

Palabras clave

Complejidad booleana y de circuitos, P vs. NP, clase $\mathsf{P}_{/\mathsf{poly}}$, endogamia de circuitos, métricas de repetitividad, diagramas de decisión binarios, cotas inferiores de circuitos, forma de circuito alternado, circuitos monótonos, técnica de limitación de anchura.

Abstract

The problem of identifying Boolean functions that require large circuits to be computed has seen little progress since its conception 70 years ago. In this work we propose new concepts and techniques that could help in this quest. Among them, we present endogamy as a property that polynomial circuits may possess to a greater extent. We will look for its experimental correlation with notions of repeatability on their computed functions, which we will have previously defined through decision diagrams. Finally, we will show a technique to find lower bounds on the size of circuits that makes use of the concept of monotonicity. With it we will prove a bound of $2n$ gates on the first level width of a specific type of circuit, which we believe can be increased by considering the rest of the circuit.

Keywords

Boolean and circuit complexity, P vs. NP, $\mathsf{P}_{/\mathsf{poly}}$ class, circuit endogamy, repeatability metrics, Binary Decision Diagrams, lower bounds of circuits, Alternating Circuit Form, monotone circuits, width limitation technique.

Índice general

Capítulo 1

Antecedentes, objetivos y plan de trabajo

El objetivo de este trabajo es definir propiedades y estrategias para tratar de mejorar nuestra comprensión sobre por qué ciertas funciones booleanas deben necesitar circuitos de tamaño superpolinómico con respecto al tamaño de su entrada para ser computadas.

El problema de la identificación de funciones complejas mediante circuitos, que forma parte del área de Complejidad Computacional, es de una gran dificultad técnica y sobre el que se han realizado escasos avances desde su concepción el siglo pasado.

El descubrimiento de que ciertas familias de funciones requieren circuitos que deben crecer superpolinómicamente con el tamaño del problema permitiría resolver el famoso P vs. NP, uno de los siete Problemas del Milenio seleccionados por el Instituto Clay de Matemáticas [Coo00]. Si esto fuera así, se habría demostrado que algunos problemas computacionales no se pueden *resolver* rápidamente a pesar de poderse *comprobar* rápidamente.

La opción contraria, es decir, que todos los problemas computacionales puedan resolverse rápidamente siempre que se puedan comprobar rápidamente, tampoco se ha descartado por ahora. Y, de hecho, probar dicha condición podría tener grandes repercusiones sobre nuestra concepción del mundo tal y como lo concebimos.

La clase NP captura muchas tareas intelectuales para las que es sencillo comprobar si están resueltas; en concreto, suele relacionarse intuitivamente con problemas que se solucionan por medio de cierta *creatividad*. La clase P se refiere a problemas resolubles rápidamente.

Por ejemplo, probar un teorema matemático puede llevar siglos de trabajo, pero una vez obtenida la demostración, en general nos es más fácil entenderla y verificar si es correcta. Si $\mathsf{P} = \mathsf{NP}$, es factible que tareas como esta pudieran resolverse y automatizarse de manera eficiente [Wig07].

1.1. Definiciones básicas

A continuación vamos a exponer las bases teóricas y antecedentes que sustentan la materia. En primer lugar trataremos con problemas de decisión y con el tiempo que tardan en computarse. Necesitamos definir para ello un modelo de computación y, además, una medida de complejidad sobre este modelo.

Definición 1.1.1. Un **problema de decisión** es un lenguaje o subconjunto de secuencias binarias de cualquier longitud: $L_g \subseteq \mathbb{Z}_2^*$. El problema también puede identificarse con su función característica o indicatriz [AB09]:

$$g : \mathbb{Z}_2^* \to \mathbb{Z}_2 \text{ tal que } g(x) = 1 \iff x \in L_g$$

Lo que nos será más útil es considerar g como una familia de funciones booleanas g_n dependientes de un tamaño de entrada n concreto:

$$g \equiv \{g_n : \mathbb{Z}_2^n \to \mathbb{Z}_2\}_{n=0}^{\infty} \text{ tal que } g(x) = 1 \iff g_{|x|}(x) = 1$$

Definición 1.1.2. Una **máquina** o **algoritmo** A que computa un problema de decisión g es una sucesión de reglas fijas tales que, al seguirlas, se calcula el valor de $g(x)$ a partir de cualquier entrada $x \in \mathbb{Z}_2^*$.

Además, el **tiempo de cómputo** T_A del algoritmo A es el número de reglas aplicadas durante dicho proceso. En general, este valor se mide asintóticamente en función del tamaño de entrada: el algoritmo tarda un tiempo $T_A(n)$ si, para cada entrada $x_n \in \mathbb{Z}_2^n$, aplica como máximo $T_A(n)$ reglas al calcular $g(x_n)$ [AB09].

Esta noción de algoritmo fue formalizada por Alan Turing en 1936 mediante su **máquina de Turing** [Tur36], un modelo de computación de igual capacidad expresiva que la inmensa mayoría de lenguajes de programación modernos.

En nuestro caso, no nos competen los detalles de su implementación, sino la posibilidad de categorizar los problemas de decisión en función de cómo de rápido pueden computarse.

Definición 1.1.3. La **clase P** es el conjunto de problemas de decisión para los que existe un algoritmo que los computa en tiempo polinómico con respecto al tamaño de entrada.

Se considera a la clase P como el conjunto de problemas que pueden resolverse en un tiempo tratable por el ser humano [Coo00]. A esta premisa se la denomina como la tesis de Cobham-Edmonds.

Por otro lado, también necesitamos un modelo de cómputo que represente la comprobación o verificación de la solución de un problema.

Definición 1.1.4. Una máquina de Turing o algoritmo **no determinista** B que computa el problema de decisión g es un conjunto de reglas tal que, en cada paso de cómputo, puede haber más de un posible paso siguiente para computar $g(x)$ a partir de cualquier entrada $x \in \mathbb{Z}_2^*$, y consideramos que la máquina dice sí si lo hace por algún camino posible.

Definición 1.1.5. La **clase NP** es el conjunto de problemas de decisión para los que existe un algoritmo no determinista que lleva a cabo un número de pasos polinómico con respecto al tamaño de entrada.

Equivalentemente, la clase NP también se puede definir como los problemas para los que su solución es verificable en tiempo polinómico. En otras palabras, para cada instancia del problema existe un testigo de tamaño polinómico —que es una prueba o pista apropiada en cada caso— y un algoritmo que deduce la solución del problema a partir de la instancia y el testigo en tiempo polinómico.

Por ejemplo, un testigo para comprobar que 21 es un número compuesto es el par de números 3 y 7, ya que se puede comprobar en tiempo polinómico que $3 \cdot 7 = 21$ [Wig07].

La equivalencia entre ambas definiciones se demuestra usando el testigo polinómico para indicar qué pasos debe seguir el algoritmo no determinista, y convertirlo, por tanto, en uno determinista.

Observación 1. Es claro que $\mathsf{P} \subseteq \mathsf{NP}$, pues un algoritmo determinista puede usarse como algoritmo no determinista con un testigo vacío.

Conjetura 1.1.1. $\mathsf{P} = \mathsf{NP}$. Es decir, $\mathsf{P} \supseteq \mathsf{NP}$, por lo que todo problema verificable de manera eficiente también podría resolverse de manera eficiente.

Conjetura 1.1.2. $\mathsf{P} \neq \mathsf{NP}$. Es decir, $\mathsf{P} \not\supseteq \mathsf{NP}$, por lo que existiría algún problema en NP que no se pueda resolver mediante un algoritmo determinista en tiempo polinómico.

Se cree con mayor aceptación que $\mathsf{P} \neq \mathsf{NP}$ [Coo00]. Los mejores candidatos para no pertenecer a la clase P son los problemas NP-completos.

Definición 1.1.6. Un problema g puede reducirse a un problema h en tiempo polinómico —denotado por $g \leq_p h$— si existe una función Ψ de coste polinómico

entre sus entradas tal que $g(x) = 1 \iff h(\Psi(x)) = 1$ [AB09].

Si tenemos un algoritmo polinómico que resuelve h, podemos aplicar Ψ a su entrada y obtener un algoritmo polinómico para g. Por tanto, h es más difícil que g.

Definición 1.1.7. Un problema h es NP-completo si pertenece a NP y todo problema $g \in \mathsf{NP}$ puede reducirse a él.

Los problemas NP-completos forman el conjunto de problemas más difíciles dentro de su clase, porque con solamente uno de ellos podemos simular el resto de estos problemas en un tiempo parecido.

1.2. Técnicas para resolver P $\neq$ NP

En los últimos 50 años, los procedimientos usados para resolver la conjetura $\mathsf{P} \neq \mathsf{NP}$ se han relacionado, principalmente, con dos tipos de técnicas demostrativas: la diagonalización y el estudio de los circuitos booleanos.

Una de las técnicas más habituales para probar teoremas en el ámbito de la Complejidad Computacional es el de la **diagonalización**. Tiene sus orígenes en la prueba de Cantor de 1891, cuando probó que el cardinal de $\mathbb{R}$ es mayor que el de $\mathbb{N}$ [Can91]. También lo usó Turing para mostrar que, con su modelo de computación, el problema de la Parada es indecidible [Tur36].

El argumento se basa en la existencia de una máquina Universal U que recibe otras máquinas como entrada y que es capaz de simularlas. Para probar que no existe un algoritmo que haga una acción A, suponemos que existe y lo combinamos con U para formar una nueva máquina U_A. La máquina U_A no hará A cuando la máquina que le pasemos como entrada sí lo haga, y viceversa. Al introducir U_A en sí misma, tendrá que hacer A cuando no lo haga, creándose así una contradicción.

Actualmente se sabe que las técnicas de diagonalización no son suficientes por sí solas para demostrar que $\mathsf{P} \neq \mathsf{NP}$. En 1975, Baker, Gill y Solovay [BGS75] probaron que la *relativización*, común a muchos de los resultados que usan estas técnicas, no es suficiente para demostrar la conjetura.

La segunda técnica es el estudio de la **complejidad de circuitos booleanos**, que es la que nosotros pretendemos utilizar. Al contrario que en los argumentos diagonales, que tratan los algoritmos como cajas negras, los circuitos permiten razonar sobre lo que ocurre dentro de los algoritmos a nivel local.

Definición 1.2.1. Un **circuito booleano** de $n \in \mathbb{N}$ entradas es un multigrafo dirigido y acíclico donde n de los nodos —o vértices— son fuentes y el resto son puertas booleanas. Los nodos fuente no tienen aristas de entrada, y consideramos que cada nodo puerta tiene exactamente dos aristas de entrada salvo para la puerta NOT, que tiene una. El tamaño del circuito es su cantidad de vértices [AB09].

El conjunto de puertas que conforman los vértices puede variar. En general, consideramos que las puertas son AND, OR y NOT. También escogeremos considerar circuitos con todas sus puertas NAND, ya que mantienen la misma expresividad.

Además, a cada nodo fuente lo identificaremos con una coordenada o variable de entrada x_i, con $0 \leq i < n$. Si consideramos que la variable está negada, la denotaremos por $\overline{x_i}$.

Definición 1.2.2. Un circuito de n entradas *computa* la función $f_n : \mathbb{Z}_2^n \to \mathbb{Z}_2$ si existe una puerta S, a la que llamaremos salida, tal que al introducir cualquier entrada $x \in \mathbb{Z}_2^n$ en los vértices fuente y aplicar la función lógica correspondiente a cada puerta en el sentido de las aristas, el valor de S es $f_n(x)$.

El conjunto de circuitos booleanos forma un modelo de computación **no uniforme**, ya que el algoritmo a computar depende del tamaño de la entrada. Este modelo tiene una clase similar a la clase P.

Definición 1.2.3. La **clase $\mathsf{P}_{/\mathsf{Poly}}$** está formada por los problemas de decisión $f \equiv \{f_n : \mathbb{Z}_2^n \to \mathbb{Z}_2\}_{n=0}^{\infty}$ para los que existe una sucesión de circuitos $\{C_n\}_{n=0}^{\infty}$, de tamaño polinómico con respecto a n, tales que C_n computa f_n.

Observación 2. Se puede demostrar que $\mathsf{P} \subset \mathsf{P}_{/\mathsf{poly}}$ usando una construcción similar a la que demuestra el teorema de Cook [Coo71]. Sin embargo, hay problemas en $\mathsf{P}_{/\mathsf{poly}}$ que no están en P, y ni siquiera en NP.

Todo lenguaje unario —$L \subseteq \{1^n : n \in \mathbb{N}\}$— está en $\mathsf{P}_{/\mathsf{poly}}$, pues puede computarse con un simple AND de sus entradas en los casos positivos, y con un circuito constantemente falso en los negativos. Pero el problema de Parada unario —$\{1^n :$ *la expansión binaria de* n *forma el par* (M, x) *tal que* M *para con la entrada* $x\}$— es indecidible, así que no está en P ni NP [AB09].

Observación 3. Si se llega a probar que existe un problema $Q \in \mathsf{NP}$ tal que $Q \notin \mathsf{P}_{/\mathsf{poly}}$, entonces $Q \notin \mathsf{P}$, luego se habría demostrado $\mathsf{P} \neq \mathsf{NP}$. Basta, por ejemplo, con encontrar alguna propiedad que todo problema en $\mathsf{P}_{/\mathsf{poly}}$ deba cumplir, y después probar que algún problema en NP no la cumple. Esta será la estrategia que seguiremos en este trabajo.

En general, se espera que los problemas NP-completos no pertenezcan a $\mathsf{P}_{/\mathsf{poly}}$.

Karp y Lipton probaron en 1982 que, si no fuera así, la jerarquía polinómica colapsaría al segundo nivel ($\mathsf{PH} = \Sigma_2^{\mathsf{P}}$) [KL82], y esto se cree altamente improbable.

Ya se han conseguido resultados parciales en el estudio de los circuitos booleanos. Para circuitos monótonos —sin puertas NOT, solo OR y AND—, la familia de circuitos mínimos que computa el problema NP-completo CLIQUE es exponencial [AB87]. Para circuitos de profundidad constante (clase NC^0), el problema PARIDAD en Pno se puede computar, y tampoco si se consideran puertas con cualquier número de aristas de entrada (clase AC^0) [AB09].

Sin embargo, aún se considera un problema abierto encontrar cotas inferiores no triviales para circuitos de carácter general. Gracias a Razborov y Rudich, lo que sí se sabe es que, si existiera una prueba así, debería *evitar* ser una **prueba natural** [RR97].

Definición 1.2.4. Una prueba natural consiste en identificar cierta propiedad que incumplan todas las funciones en $\mathsf{P}_{/\mathsf{poly}}$ tal que, para cada tamaño, se cumpla para una gran proporción de funciones y, además, se compruebe en tiempo polinómico con respecto a la tabla de verdad de cada función —es decir, 2^n.

Razborov y Rudich mostraron que la existencia de propiedades así incumpliría ciertas suposiciones estándar. Se espera entonces que los factores que nos ayuden a determinar la no pertenencia a $\mathsf{P}_{/\mathsf{poly}}$ o bien se cumplan en una proporción escasa de funciones, o bien sean muy costosos de computar.

1.3. Plan de trabajo

Una vez hemos descrito los antecedentes de donde proceden las cuestiones que intentamos responder en este texto, será en los próximos capítulos donde propondremos y desarrollaremos nuestras principales estrategias.

En primer lugar, creemos que las funciones más repetitivas son las que pueden computarse por circuitos más pequeños en número de puertas. En el capítulo 2 trataremos de capturar la intuición de *repetitividad* a través de distintas métricas de desorden. Llegaremos con ellas al concepto de Diagrama de Decisión Binario, que es ampliamente conocido y utilizado como estructura de datos para almacenar funciones booleanas.

En el trabajo de fin de grado asociado al Grado en Ingeniería Informática [RM22a] se han usado estas métricas para clasificar un conjunto de cuatro millones de funciones booleanas provenientes de un trabajo anterior [VE21]. Con ellas se ha conseguido una predicción del 95,29% de acierto, por lo que creemos que consiguen

capturar gran parte de los motivos por los que las funciones pueden computarse con circuitos pequeños.

En el capítulo 3 presentaremos el concepto de endogamia de un circuito, que se trató por primera vez en otro trabajo previo [RC20] y que calcula en qué medida el circuito repite parientes si lo interpretamos como el árbol genealógico de su nodo de salida. Lo relacionaremos con la invariancia o equidad de variables, y, en especial, con la repetitividad previamente definida. Propondremos conceptos puente que las unan y expliquen su mecánica y trataremos de reforzar nuestras hipótesis mediante experimentos de correlación entre las nociones descritas.

A continuación, en el capítulo 4, estudiaremos los circuitos booleanos que toman una forma concreta: la Forma de Circuito Alternado (AC), que alterna niveles de puertas AND y OR y solo tiene puertas NOT en su primer nivel. Probaremos que todo circuito puede transformarse polinómicamente en uno equivalente con esta forma. Después conseguiremos demostrar algunas cotas inferiores en anchura para su primer nivel de puertas.

En un circuito con anchura necesariamente limitada —como lo son los circuitos de tamaño polinómico—, la endogamia interna es inevitable. En sus niveles más bajos, un circuito así puede formar exhaustivamente todas las combinaciones de unos pocos bits. En alturas superiores, dicha exhaustividad es imposible para combinaciones de muchos más bits debido a la limitación de anchura. Los circuitos pueden intercambiar cómputos en anchura por cómputos en altura, pero la necesaria endogamia del circuito dificultará alcanzar dicha exhaustividad por medio de tal intercambio: la endogamia obligará a los niveles superiores a privilegiar enormemente su atención a algunas combinaciones de bits sobre otras, sesgando los cómputos que pueden construir.

Nuestra técnica para construir funciones que requieran circuitos con cierta anchura se basa, precisamente, en obligar al circuito a combinar exhaustivamente ciertos bits. Es por ello que pensamos que, para llevar la técnica al siguiente nivel y permitirle hallar cotas sobre el número total de puertas en cualquier altura, necesitamos integrar la mecánica de la endogamia en nuestros razonamientos, al ser lo que impedirá la exhaustividad en niveles más altos. Nuestros experimentos del capítulo 3 para entender cómo funciona la endogamia constituyen nuestro primer paso para tratar de integrarla en nuestra técnica desarrollada en el capítulo 4.

Desde 1983, la cota inferior máxima conocida del tamaño de un circuito general que computa problemas en NP era de $3n - o(n)$ puertas [Blu83]. En 2016 se consiguió aumentar a $\left(3 + \frac{1}{86}\right) n - o(n)$ [FGHK16] y muy recientemente en 2022

se ha llegado a $3.1n - o(n)$ [LY22]. Todas estas cotas están basadas en una técnica llamada *eliminación de puertas*. Sin embargo, esta técnica jamás podrá obtener cotas superlineales [GHKK18].

A partir de nuestro trabajo, esperamos que la técnica propuesta pueda utilizarse en el futuro para intentar probar cotas inferiores más altas de lo que se ha conseguido hasta ahora para problemas en NP; y en particular, que pueda permitir cotas superlineales, o, idealmente, superpolinómicas.

Capítulo 2

Métricas de repetitividad

En este capítulo, tratamos de averiguar qué factores influyen para que una familia de funciones $\{f_n : \mathbb{Z}_2^n \to \mathbb{Z}_2\}_{n=0}^{\infty}$ necesite ser computada por circuitos cuyos tamaños crezcan más allá de lo polinómico.

Hemos conjeturado que, si cada f_n presenta una alta repetitividad, los circuitos que las computan podrán permitirse un menor número de puertas. Y al contrario, si conseguimos medir una baja repetitividad, o una gran cantidad de desorden, los circuitos necesitarán una gran cantidad de puertas para poder computar dichas funciones.

Debemos formalizar el concepto de repetitividad, y crear así métricas que evalúen la sencillez o complejidad de cada función en base a esta intuición. Las definiciones de estas métricas forman parte del trabajo de fin de grado asociado al Grado en Ingeniería Informática [RM22a], pero es en el presente texto donde se demuestran sus propiedades y relaciones entre ellas, así como las afirmaciones no probadas en el texto previo.

2.1. Métrica Chunks-k

Para poder medir valores numéricos sobre una función booleana, tenemos que expresarla mediante alguna representación concreta sobre la que calcularlos.

La representación más directa es a través de su tabla de verdad, a cuya fila de resultados llamaremos su cadena.

Definición 2.1.1. Para una función $f_n : \mathbb{Z}_2^n \to \mathbb{Z}_2$ de n entradas booleanas, su **representación en cadena** es el vector fila $chain(f_n)$, cuyas componentes son todas las imágenes de la función para cada una de sus posibles entradas. En

concreto, para todo $0 \leq i < 2^n$, su componente i es $f_n(\text{bin}(i))$, donde $\text{bin}(i)$ es el número i en base 2.

Ejemplo 1. Para $n = 3$, consideremos la función paridad p_3, que toma el valor 1 si y solo si la cantidad de entradas con valor 1 es impar. Entonces $chain(p_3) = (0, 1, 1, 0, 1, 0, 0, 1) =$ '01101001'.

Esta es su tabla de verdad, donde e_0, e_1 y e_2 son las tres entradas de la función:

e_0	0	0	0	0	1	1	1	1
e_1	0	0	1	1	0	0	1	1
e_2	0	1	0	1	0	1	0	1
p_3	0	1	1	0	1	0	0	1

La primera forma de calcular la repetitividad de una función es dividir su tabla de verdad y contar cuántas subdivisiones, o *chunks* de la cadena, se repiten.

Definición 2.1.2. Dada una cadena $c_n \in \mathbb{Z}_2^{2^n}$, definimos la métrica $Chunks_k(c_n)$, con $0 \leq k \leq n$, como el número de subcadenas distintas de tamaño 2^k que empiezan en las posiciones $i \cdot 2^k$.

Observación 4. El máximo valor que puede devolver la métrica $Chunks_k$ es 2^{n-k}, que es el número de subcadenas, de tamaño 2^k, en las que hemos dividido la cadena original, de tamaño 2^n.

Teorema 2.1.1. Existe una familia de funciones $\{s_j\}_{j=0}^{\infty}$ para las que existen circuitos polinómicos —con respecto al tamaño de su entrada— que las computan y que, sin embargo, alcanzan el valor máximo de la métrica $Chunks_k$ para algún valor de k.

Demostración. Sea la función s_j cuya cadena es la concatenación de todos los números binarios de tamaño 2^j en orden creciente (es decir, del 0 al $2^{2^j} - 1$). Tiene longitud 2^{2^j+j}, luego s_j recibe como entrada $n = 2^j + j$ valores.

Vamos a probar que s_j puede calcularse en tiempo polinómico. Dada una entrada i, el bit de salida $s_j(i)$ se encuentra en la posición $x \coloneqq i \,\%2^j$ del número binario $y \coloneqq i//2^j$. La operación $\%$ es el resto y $//$ es la división entera entre los dos números. Ambas operaciones se pueden calcular en tiempo polinómico.

El bit en la posición x del número y en binario es $b \coloneqq (y//2^x)\,\%2$. De nuevo, se puede calcular en tiempo polinómico. Como $s_j(i) = b$, hemos probado que existe un algoritmo polinómico que lo calcula. Como la clase P pertenece a $\mathsf{P}_{/\mathsf{poly}}$, también existe un circuito de tamaño polinómico que lo calcula.

Queda por probar que $Chunks_j(chain(s_j)) = 2^{n-j} = 2^{2^j}$. Pero esto es inmediato: por construcción de $chain(s_j)$, cada subdivisión de tamaño 2^j es un número binario distinto al resto. □

Corolario 2.1.1.1. La métrica $Chunks_k$ no captura correctamente el crecimiento polinómico de algunas funciones sencillas, luego no es el factor determinante que distingue qué funciones requieren circuitos pequeños o grandes.

2.2. Métrica Concat-Times

La segunda fórmula que pretende capturar la repetitividad de las funciones sencillas se basa en dividir la tabla de verdad por la mitad, de manera recursiva, para contar cuántas veces se obtienen dos partes distintas. Las subcadenas iguales se agrupan en potencias de 2 (TIMES) y las distintas se concatenan (CONCAT), por lo que se llamará CONCAT-TIMES.

La nueva representación de las funciones tendrá forma de árbol, que refleja el carácter recursivo de su definición. Usaremos para ello el árbol de derivación de las expresiones generadas por una gramática independiente del contexto.

Definición 2.2.1. La **gramática CONCAT-TIMES** es la tupla $\mathcal{G} = (V, A, P, S)$, donde:

- $A = \{0, 1, (,), +, 2^m\cdot\}$, con $m \in \mathbb{N}$, es el alfabeto o conjunto de terminales.
- $V = \{S, C\}$ es el conjunto de variables, con S la variable inicial.
- El conjunto de producciones, P, es:

$$S \to 2^m \cdot C$$
$$C \to 0 \mid 1 \mid (S + S)$$

El siguiente paso es dar un significado a cada producción de la gramática. Es decir, necesitamos traducir expresiones sintácticas de $\mathcal{G}$ —palabras del lenguaje $L(\mathcal{G})$— en una cadena de 0 y 1, de longitud potencia de 2, que podamos asociar a una cadena c_n.

Primeramente identificamos el símbolo $m \in \mathbb{N}$ con la semántica habitual de los numerales. Queda por determinar el significado de la variable inicial S, que definiremos de manera recursiva y mediante una función auxiliar para definir las producciones de C.

Además, tomamos la concatenación de las cadenas c_1 y c_2 como c_1c_2, y la concatenación de c consigo misma k veces como c^k.

Definición 2.2.2. Definimos la **función semántica** $\mathcal{S}$, que transforma palabras de $L(\mathcal{G})$ en cadenas de $\mathbb{Z}_2^{2^k}$ para algún $k \in \mathbb{N}$, de la siguiente forma:

$$\begin{aligned} \mathcal{S}[\![2^m \cdot C]\!] &\coloneqq \mathcal{S}_C[\![C]\!]^{2^m} \\ \mathcal{S}_C[\![0]\!] &\coloneqq \text{`0'} \\ \mathcal{S}_C[\![1]\!] &\coloneqq \text{`1'} \\ \mathcal{S}_C[\![(S_1 + S_2)]\!] &\coloneqq \mathcal{S}[\![S_1]\!]\mathcal{S}[\![S_2]\!] \end{aligned}$$

Ejemplo 2. Sea la función cuya cadena es $c =$ '1110'. Hay varias expresiones en $L(\mathcal{G})$ cuya semántica se corresponde con c:

$$2^0 \cdot (2^1 \cdot 1 + 2^0 \cdot (2^0 \cdot 1 + 2^0 \cdot 0)) \simeq (\text{`11'} + (\text{`1'} + \text{`0'}))$$

$$2^0 \cdot (2^0 \cdot 1 + 2^0 \cdot (2^1 \cdot 1 + 2^0 \cdot 0)) \simeq (\text{`1'} + (\text{`11'} + \text{`0'}))$$

$$2^0 \cdot (2^0 \cdot (2^0 \cdot 1 + 2^0 \cdot 1) + 2^0 \cdot (2^0 \cdot 1 + 2^0 \cdot 0)) \simeq ((\text{`1'} + \text{`1'}) + (\text{`1'} + \text{`0'}))$$

De las tres expresiones solamente nos interesa la primera, pues concatena subcadenas solo si tienen la misma longitud, y agrupa las subcadenas iguales entre sí. Por tanto, estamos trabajando con un subconjunto reducido de $L(\mathcal{G})$, que denotaremos por $L_r(\mathcal{G})$.

Necesitamos una manera de traducir unívocamente cadenas en expresiones de $L(\mathcal{G})$. Es decir, vamos a definir la inversa de $\mathcal{S}$. Con el ejemplo 2 hemos observado que esta función no va a ser sobreyectiva, y que, por definición, se tendrá que $L_r(\mathcal{G}) \coloneqq \text{im}(\mathcal{S}^{-1})$.

Definición 2.2.3. La inversa de $\mathcal{S}$, $\mathcal{S}^{-1}$, traduce una cadena c_n, de longitud 2^n, en una palabra de $L_r(\mathcal{G})$. La definimos por inducción en n:

- Para el caso base de $k = 0$:

$$\mathcal{S}^{-1}[\![\text{`0'}]\!] \coloneqq 2^0 \cdot 0 \qquad \mathcal{S}^{-1}[\![\text{`1'}]\!] \coloneqq 2^0 \cdot 1$$

- Para $k > 0$, sea c de longitud 2^k. La partimos por la mitad: $c = c_1c_2$ con c_1 y c_2, ambas de longitud 2^{k-1}.
  - Si $c_1 \neq c_2$, entonces $\mathcal{S}^{-1}[\![c]\!] \coloneqq 2^0 \cdot (\mathcal{S}^{-1}[\![c_1]\!] + \mathcal{S}^{-1}[\![c_2]\!])$. Es decir, las dos subcadenas no se repiten entre sí.
  - Si son iguales, tenemos por hipótesis que $\mathcal{S}^{-1}[\![c_1]\!] = 2^m \cdot C$ para algún $C \in L_r(\mathcal{G})$. Entonces definimos $\mathcal{S}^{-1}[\![c]\!] \coloneqq 2^{m+1} \cdot C$. Es decir, la subcadena C se repite 2^{m+1} veces.

Teorema 2.2.1. $\mathcal{S}^{-1}$ es inyectiva.

Demostración. Primero, se tiene que $\mathcal{S}[\![\mathcal{S}^{-1}[\![c]\!]]\!] = c$. Es directo para el caso $k = 0$.

Para $k > 0$, sea $c = c_1c_2$ y $\mathcal{S}^{-1}[\![c_1]\!] = 2^m \cdot C$. Por hipótesis, se tiene:

$$c_1 = \mathcal{S}[\![\mathcal{S}^{-1}[\![c_1]\!]]\!] = \mathcal{S}[\![2^m \cdot C]\!] = \mathcal{S}_C[\![C]\!]^{2^m}$$

Luego, si $c_1 = c_2$:

$$\mathcal{S}[\![\mathcal{S}^{-1}[\![c_1c_1]\!]]\!] = \mathcal{S}[\![2^{m+1} \cdot C]\!] = \mathcal{S}_C[\![C]\!]^{2^{m+1}} = \mathcal{S}_C[\![C]\!]^{2^m}\mathcal{S}_C[\![C]\!]^{2^m} = c_1c_1$$

Y si $c_1 \neq c_2$:

$$\mathcal{S}[\![\mathcal{S}^{-1}[\![c_1c_2]\!]]\!] = \mathcal{S}[\![2^0 \cdot (\mathcal{S}^{-1}c_1]\!] + \mathcal{S}^{-1}[\![c_2]\!])]\!] = \mathcal{S}[\![\mathcal{S}^{-1}[\![c_1]\!]]\!]\mathcal{S}[\![\mathcal{S}^{-1}[\![c_2]\!]]\!] = c_1c_2$$

Por último, $\mathcal{S}^{-1}$ es inyectiva, porque si $\mathcal{S}^{-1}[\![c_a]\!] = \mathcal{S}^{-1}[\![c_b]\!]$ se da que:

$$c_a = \mathcal{S}[\![\mathcal{S}^{-1}[\![c_a]\!]]\!] = \mathcal{S}[\![\mathcal{S}^{-1}[\![c_b]\!]]\!] = c_b$$

□

Observación 5. La función $\mathcal{S}^{-1}$ provee una forma concreta de generar una cadena. Si esta fuera la forma óptima, supondría poder calcular la métrica que definamos a partir de ella en un tiempo polinómico. Se estaría así cumpliendo la 1ª condición para que sea una *natural proof*. Con una alta seguridad también se cumple la 2ª, pues la propiedad se cumple en una proporción no despreciable de funciones de cada tamaño.

Por tanto, y como ya indicamos en el capítulo 1, una métrica solamente basada en este procedimiento no permitirá encontrar funciones superpolinómicas bajo hipótesis estándar. Aun así, creemos que la búsqueda del verdadero factor que determina la necesidad de crecimiento superpolinómico puede pasar por factores que no basten por sí solos, siempre y cuando nos guíen hacia otros mejores.

En todo caso, nos queda por encontrar el árbol que represente cada función, del que obtendremos nuestra siguiente métrica de repetitividad.

Teorema 2.2.2. La gramática $\mathcal{G}$ no es ambigua porque es una gramática $\mathsf{LL}(1)$.

Demostración. Una gramática es $\mathsf{LL}(1)$ si y solo si no genera conflictos FIRST/FIRST ni conflictos FIRST/FOLLOW. A continuación comprobamos ambas afirmaciones:

- $\mathcal{G}$ no tiene conflictos FIRST/FIRST porque cada producción comienza con un terminal distinto: $2^m\cdot$, 0, 1 y (.

- $\mathcal{G}$ no tiene conflictos FIRST/FOLLOW porque no genera la cadena vacía ϵ.

□

Corolario 2.2.2.1. Que $\mathcal{G}$ no sea ambigua implica que existe un único árbol de derivación para cada expresión sintáctica en $L_r(\mathcal{G})$. A este árbol de derivación lo llamamos el **árbol CONCAT-TIMES**, y lo usaremos como una representación más de las funciones booleanas.

En general, identificamos una función booleana por su cadena, que a su vez se identifica por su expresión sintáctica en $\mathcal{G}$, que a su vez se identifica por su árbol de derivación.

$$\text{Función booleana} \xleftrightarrow{chain} \text{Cadena o tabla de verdad} \xleftrightarrow{\mathcal{S},\mathcal{S}^{-1}} \text{Expresión sintáctica en } L_r(\mathcal{G}) \xleftrightarrow{\Rightarrow^*} \text{Árbol de derivación}$$

A partir del árbol CONCAT-TIMES podemos definir la métrica que estamos buscando.

Definición 2.2.4. La **métrica CONCAT-TIMES**, o CT, de la cadena de una función f, es el número de hojas con 0 o 1 de su árbol CONCAT-TIMES. Este árbol es, a su vez, el árbol de derivación de la expresión generada por $\mathcal{G}$ cuya semántica es la cadena asociada a f.

Esta métrica es equivalente al número de veces (más uno) que la cadena no se ha podido subdividir en dos partes iguales entre sí. Es decir, las veces en las que su expresión sintáctica asociada se ha derivado por la producción $C \to (S + S)$ (más uno), o el número de hojas (más uno) con el terminal $+$.

Ejemplo 3. Sea $c =$ '00011011' la cadena de la función s_1 considerada en el resultado 2.1.1. En la figura 2.1 está su árbol CONCAT-TIMES, y con él obtenemos que $CT(c) = 6$.

Ejemplo 4. Sea $c =$ '01101001' la cadena de la función paridad p_3, considerada en el ejemplo 1. En la figura 2.2 se encuentra su árbol CONCAT-TIMES.

El valor de la métrica $CT(chain(p_3))$ es el máximo posible: hay una hoja por cada posible entrada de la función. Vamos a probar que se llega también al máximo para la función paridad p_n con n arbitrario, y que, sin embargo, es una familia polinómica.

Teorema 2.2.3. Sea $\{p_n : \mathbb{Z}_2^n \to \mathbb{Z}_2\}_{n=0}^{\infty}$ el problema Paridad: $p_n(x) = 1$ si y solo si tiene un número impar de entradas con valor 1. El problema Paridad pertenece a la clase $\mathsf{P}_{/\mathsf{poly}}$.

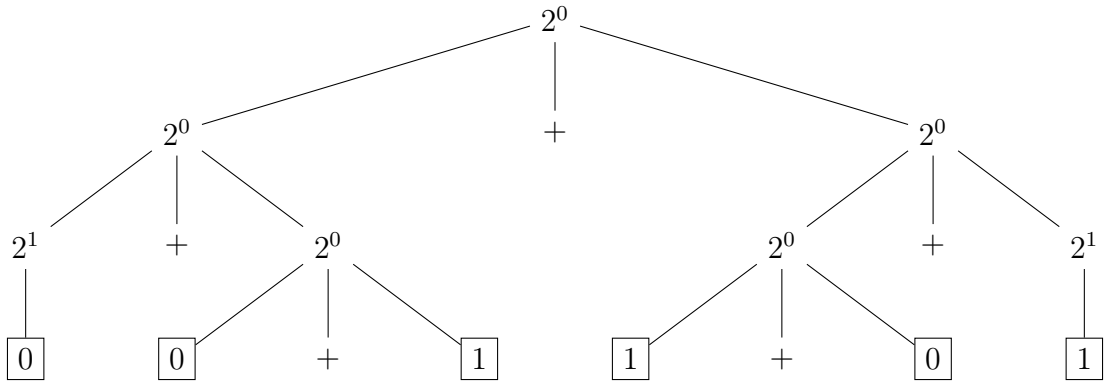

Figura 2.1: Árbol CONCAT-TIMES de la función s_1.

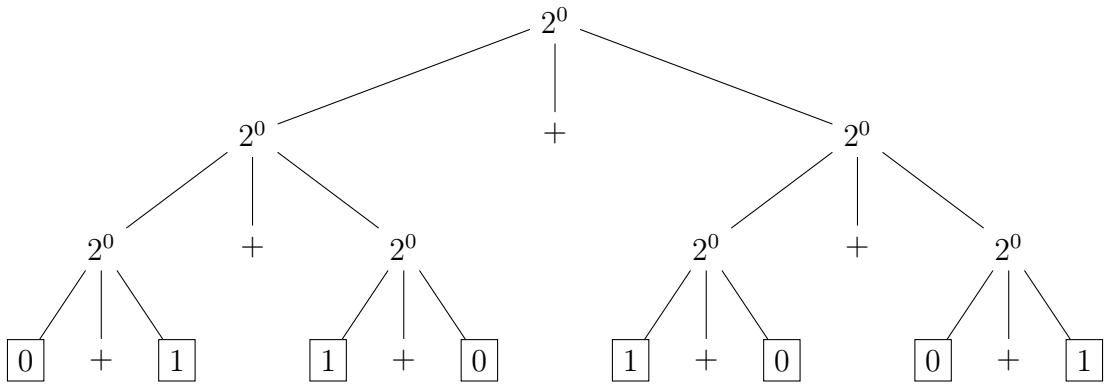

Figura 2.2: Árbol CONCAT-TIMES de la función p_3 del ejemplo 4.

Demostración. En efecto, p_n puede calcularse con un circuito de $n-1$ puertas XOR que conecten todas las entradas una a una. □

Teorema 2.2.4. $CT(chain(p_n)) = 2^n$.

Demostración. Lo probamos por inducción en n. Ya tenemos el caso base de $n = 3$, y es fácil comprobarlo para los casos inferiores.

Para $k > 3$, si $chain(p_{k-1}) = c$, entonces $chain(p_k) = c\ \overline{c}$. Esto es porque el número de entradas a 1 cuando $x_0 = 0$ es uno menos —y, por tanto, de paridad contraria— al número de entradas a 1 cuando $x_0 = 1$.

Por tanto, $\mathcal{S}^{-1}[\![c\ \overline{c}]\!] = 2^0 \cdot (\mathcal{S}^{-1}[\![c]\!] + \mathcal{S}^{-1}[\![\overline{c}]\!])$, ya que nunca son subcadenas iguales. El número de hojas 0 o 1 será la suma de las hojas de ambos lados, y, por hipótesis de inducción, $CT(c) = 2^{k-1}$, luego $CT(c\ \overline{c}) = 2^k$. □

La noción de repetitividad de una función no debería estar ligada al orden en el que disponemos sus variables en la tabla de verdad. Por ejemplo, en la figura 2.3 se encuentran las tablas correspondientes a las permutaciones $(0, 1, 2)$ y $(2, 1, 0)$ de las entradas de la función s_1. Mientras que la métrica CT de la primera cadena, como ya vimos en el ejemplo 3, da 6, la segunda cadena aprovecha mejor los patrones de repetitividad no consecutivos y rebaja el valor a 4. En general, hay una posible cadena por cada permutación σ de las n entradas.

e_0	0	0	0	0	1	1	1	1
e_1	0	0	1	1	0	0	1	1
e_2	0	1	0	1	0	1	0	1
s_1	0	0	0	1	1	0	1	1

e_2	0	1	0	1	0	1	0	1
e_1	0	0	1	1	0	0	1	1
e_0	0	0	0	0	1	1	1	1
s_1	0	1	0	1	0	0	1	1

Figura 2.3: Tablas de verdad de s_1 con permutaciones de entradas $(0, 1, 2)$ y $(2, 1, 0)$.

Definición 2.2.5. Sea una permutación $\sigma \in S_n$ (es decir, del grupo de permutaciones de n elementos). La **σ-cadena** de f_n, σ-$chain(f_n)$, es el vector fila cuya componente i es $f_n(\sigma(\text{bin}(i)))$.

Observación 6. Al poder permutar las n entradas de la función, que se corresponden con las n posiciones de bits de $\text{bin}(i)$, se obtienen $n!$ posibles σ-cadenas asociadas a una función de n entradas.

Habiendo visto que las métricas definidas hasta ahora dan valores muy distintos para distintas permutaciones de una misma tabla de verdad, necesitamos que el concepto de repetitividad sea más homogéneo. Podemos conseguirlo si tomamos el mínimo de los valores obtenidos sobre todas las σ-cadenas.

Definición 2.2.6. Dada una métrica sobre cadenas $m : \mathbb{Z}_2^n \to \mathbb{Z}$, su **métrica invariante asociada**, eq_m, es:

$$\text{eq}_m(c_n) = \min_{\sigma \in S_n} m(\sigma\text{-}chain(c_n))$$

Aunque esta técnica es útil, nos conviene más poder reflejar la invariancia ante permutaciones de entradas dentro de las propias definiciones de repetitividad, y no mediante un artificio externo.

2.3. Métrica Cube

Una forma de representar una función f que permita simetrías ante permutaciones de sus n entradas es a través del espacio n-dimensional. En $\mathbb{R}^n$, el conjunto de puntos con coordenadas $x = (x_0, \ldots, x_{n-1}) \in \mathbb{Z}_2^n$ toma la forma de un hipercubo de lado 1.

Definición 2.3.1. Para una función $f_n : \mathbb{Z}_2^n \to \mathbb{Z}_2$, su **representación en hipercubo** son los vértices valorados del hipercubo H^n, que numeramos del 0 al $2^n - 1$. El valor del vértice $i \in H^n$, que tiene como coordenadas $\text{bin}(i)$, es el valor $f_n(\text{bin}(i))$.

Ejemplo 5. Sea f la función tal que $chain(f) =$ '01011101'. En la figura 2.4 está la representación de f como cubo en 3 dimensiones. También está representado el recorrido de vértices que forma la cadena $chain(f)$ sobre el cubo, que empieza en el vértice azul y acaba en el rojo.

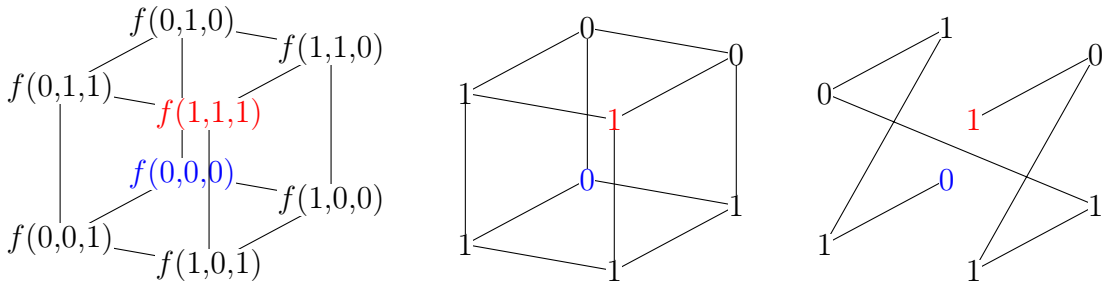

Figura 2.4: Representación de hipercubo de la función f del ejemplo 5.

Observación 7. Podemos considerar otros recorridos (σ-cadenas) de vértices que se correspondan con las $n!$ distintas permutaciones de las entradas de la función. Escoger la entrada en la posición i es equivalente a escoger la cara $(n - i - 1)$-dimensional del hipercubo que toca recorrer en ese orden.

En el ejemplo 5, entre las caras lateral ($x = 0$), inferior ($y = 0$), y trasera ($z = 0$) se escoge la lateral; entre el eje $y = 0$ y el $z = 0$ se escoge el $y = 0$, y así queda totalmente determinada su trayectoria.

En la figura 2.5 se muestran todas las σ-cadenas posibles para la función f del ejemplo 5. Las dos últimas opciones, que escogen primero separar por la variable z, consiguen agrupar toda una cara constantemente a 1. Esta parece una elección más acertada que las otras para encontrar su repetitividad.

Nuestra próxima métrica divide el hipercubo en dos mitades, igual que la métrica CT, pero por aquella dimensión en la que más repetitividad encuentre, aquella en la que minimice su valor.

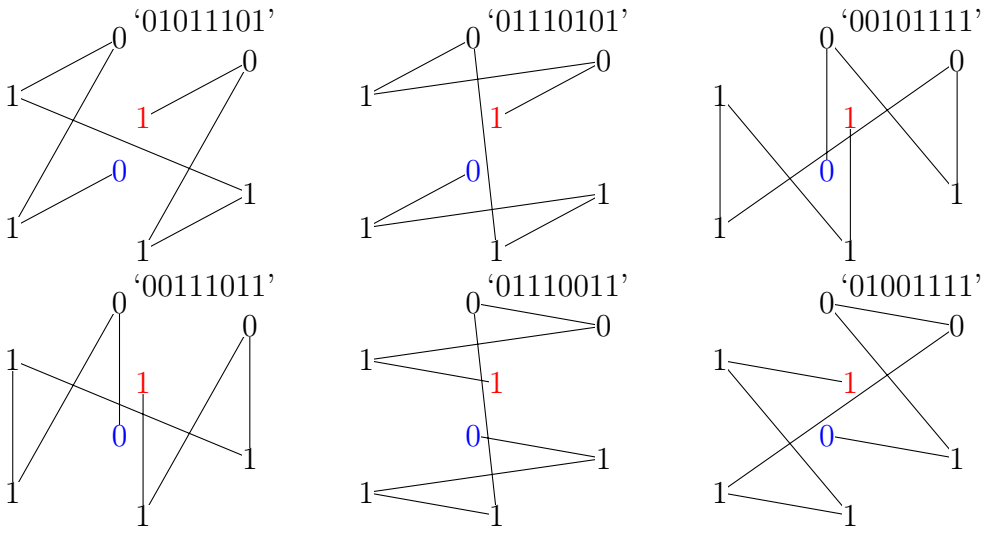

Figura 2.5: Todas las representaciones en cubo equivalentes de f en función del orden de sus entradas.

Definición 2.3.2. Dada una función $f_n : \mathbb{Z}_2^n \to \mathbb{Z}_2$, y su representación en cubo H^n, se define la métrica $Cube(H^n)$ de la siguiente manera:

- $Cube(\{0\}) = Cube(\{1\}) = 1$.
- Para todo $k \in \{1, ..., n\}$ consideramos el hipercubo H^k. Para cada una de sus dimensiones $d \in \{0, \ldots, k-1\}$, definimos los dos subcubos de dimensión $k-1$ obtenidos al proyectar H^k sobre la coordenada d, fijada a 0 y 1, respectivamente.

$$H_{d,0}^{k-1} \coloneqq H^k \cap \{x_d = 0\}$$
$$H_{d,1}^{k-1} \coloneqq H^k \cap \{x_d = 1\}$$

  Definimos el valor C_d^k como:

  - Si, al proyectar sobre la dimensión d, los dos subcubos coinciden en valor (es decir, $H_{d,0}^{k-1} = H_{d,1}^{k-1}$), entonces $C_d^k \coloneqq Cube(H_{d,0}^{k-1})$.
  - Si son distintos, entonces $C_d^k \coloneqq Cube(H_{d,0}^{k-1}) + Cube(H_{d,1}^{k-1})$.

  El valor de la métrica para el hipercubo H^k será aquel que minimice C_d^k, tomando d sobre todas sus posibles dimensiones:

$$Cube(H^k) = \min_{0 \le d < k} C_d^k$$

Por tanto, para $n > 1$, $Cube(H^n) = \min_{0 \leq d < n} C_d^n$.

Teorema 2.3.1. $Cube(p_n) = 2^n$.

Véase el ejemplo de p_3 en la figura 2.6.

Demostración. Se puede probar fácilmente por inducción en n. El caso $k = 1$ es trivial, pues $p_1 = \text{id}$ y $\{x_0 = 0\} \neq \{x_0 = 1\}$, luego $Cube(\text{id}) = 2$.

Sea $k > 1$. Por hipótesis de inducción, se tiene que $Cube(H_{d,0}^{k-1}) = Cube(H_{d,1}^{k-1}) = 2^{k-1}$ para todo $0 \leq d < k$.

Sin embargo, $H_{d,0}^{k-1}$ tiene las mismas coordenadas que $H_{d,1}^{k-1}$ salvo por la coordenada d, que es contraria. Por tanto, la paridad del número de coordenadas a 1 es la contraria en cada punto simétrico con respecto a la variable d.

Esto implica que ambos subcubos son distintos, y que, para todo d, se da $Cube(H^k) = Cube(H_{d,0}^{k-1}) + Cube(H_{d,1}^{k-1}) = 2^k$. □

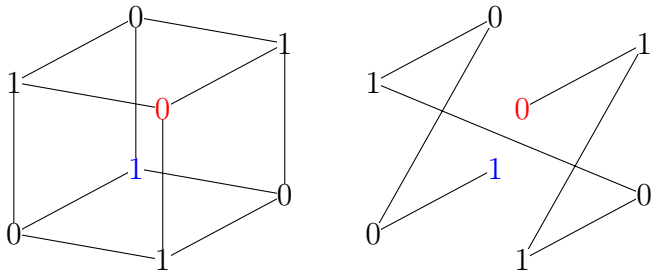

Figura 2.6: Representación de hipercubo de la función paridad p_3.

Corolario 2.3.1.1. Tanto el teorema 2.2.4 como el 2.3.1 nos demuestran que las métricas CT y $Cube$ aún no son capaces de reflejar por qué una función sencilla como p_n es computada por un circuito booleano pequeño.

2.4. Diagramas de Decisión Binarios

La estructura en forma de árbol que hemos desarrollado tanto para la métrica CONCAT-TIMES (sección 2.2) como para la métrica *Cube* (sección 2.3) se puede generalizar a un concepto ya conocido y muy útil de la informática: los diagramas de decisión binarios (*Binary Decision Diagrams*, abreviados como BDD) [BSSW10].

Definición 2.4.1. Un **diagrama de decisión binario**, o BDD, es un árbol binario con raíz cuyos nodos internos son nodos decisión y cuyas hojas son nodos salida con valor 0 o 1. Cada nodo decisión v tiene asociado una variable de entrada $i_v \in \mathbb{N}$,

y de él salen dos hijos: el nodo v_0, para el que $x_i = 0$, y el nodo v_1, para el que $x_i = 1$.

Todo nodo v representa una función $f_v : \mathbb{Z}_2^n \to \mathbb{Z}_2$. Para un valor de entrada $x \in \mathbb{Z}_2^n$:

- Si v es un nodo salida, entonces $f(x)$ es el valor constante del nodo.
- Si v es un nodo decisión con entrada asociada i, y se da que $x_i = j$, entonces el valor de la función es el valor del subárbol escogido por el nodo decisión: $f_v(x) = f_{v_j}(x)$.

Por último, se dice que un BDD representa la función booleana f si esta es la función que representa su raíz.

Definición 2.4.2. Un BDD es **s-oblivious** para una secuencia de variables $s = (s_0, \ldots, s_{\ell-1}) \in \mathbb{N}^\ell$ si todos los nodos pueden agruparse en $\ell+1$ niveles: en el nivel $i < \ell$ todos los nodos son de decisión con variable asociada s_i y en el nivel ℓ están los nodos salida. Más aún, si s corresponde a una permutación σ de las entradas $(\sigma(0), \ldots, \sigma(\ell-1))$, entonces el BDD está **ordenado** y se denota por OBDD.

Que esté ordenado es equivalente a que, en todo camino desde la raíz a las hojas, las variables asociadas a los nodos aparezcan siempre en el mismo orden.

Lee introdujo los BDD en 1959 [Lee59] y Bryant definió los OBDD por primera vez en 1986 [Bry86]. Esta estructura de datos es la más utilizada para representar las funciones booleanas, con aplicaciones tan diversas como la verificación, la optimización o los algoritmos de grafos [BSSW10].

Con nuestras propias definiciones, podemos asegurar que el árbol CONCATTIMES es, en realidad, exactamente un OBDD con $\sigma = \mathrm{id}$. Su primera decisión es mediante x_0 para dividir la tabla de verdad en dos mitades; cada mitad se subdivide con x_1, y así sucesivamente. Además, la métrica *Cube* es otro tipo de BDD, pero sin orden.

Sin embargo, no hemos considerado aún estructuras que aprovechen valores computados previamente para no malgastar el doble de recursos. A esto se le llama ser reducido.

Definición 2.4.3. Un BDD está **reducido** si es el BDD de tamaño mínimo para la función que computa.

Bryant [Bry86] probó que un BDD reducido y ordenado, también conocido como ROBDD, es único salvo isomorfismo, y que se construye a partir de la aplicación reiterada de dos reglas en orden arbitrario:

- Reutilizando subgrafos isomorfos, es decir, que computen la misma función.
- Eliminando nodos decisión cuyos dos hijos sean isomorfos, ya que, en realidad, no están tomando decisión alguna.

Además, en 1993, Sieling y Wegener [SW93] consiguieron obtener un algoritmo lineal en el tamaño del OBDD que se quiere reducir.

Los ROBDD tienen muy buenas propiedades para los usos prácticos como representación de funciones booleanas. Al suponer que el orden de las variables se mantiene, se pueden conseguir operaciones para BDD —que se traducen en operaciones para funciones, como la evaluación, la satisfactibilidad o la igualdad— de manera eficiente. Además, los ROBDD son representaciones pequeñas para la mayoría de funciones polinómicas que se utilizan, por lo que pueden ahorrar espacio.

Capítulo 3

Experimentos de endogamia

Uno de los principales motivos por los que creemos que algunas funciones no requieren circuitos superpolinómicos para ser computadas es porque pueden ser computadas por circuitos que contienen un alto nivel de *endogamia* interna.

La endogamia $\mathcal{E}$ es una propiedad de un circuito que medirá cuántos parientes comunes de cualquier nivel tiene su puerta de salida, interpretando el circuito como un árbol genealógico. Por ejemplo, si la última puerta es una NAND que comparte sus dos padres, esta puerta se comporta como una NOT, por lo que su complejidad no aumenta con respecto al nivel de puertas anterior.

Para convertir la noción de endogamia en algo puramente estructural, en este capítulo el conjunto de puertas de los circuitos estará formado únicamente por la puerta NAND.

3.1. Invariancia y equidad de variables

Supongamos que un circuito de n entradas tiene anchura ilimitada en cada nivel del circuito. En las puertas de su primer nivel podrán reunirse hasta n^2 combinaciones de dos entradas. En el siguiente nivel serán n^{2^2} combinaciones posibles de cuatro entradas, y en el i serán n^{2^i} de 2^i. Llegará un nivel ℓ en el que deberán reunirse de nuevo entre sí esos n^{2^ℓ} nodos para producir la salida del circuito, lo que requerirá una profundidad extra de $\log n^{2^\ell} = 2^\ell \log n$ niveles con $\frac{n^{2^\ell}}{2^k}$ puertas en cada nivel $\ell + k$.

En total, en los niveles previos a ℓ habrá $\sum_{i=1}^{\ell} n^{2^i}$ puertas, y otras n^{2^ℓ} en los siguientes niveles. Si ℓ se toma tal que la profundidad del circuito sea mayor que

logarítmica, la explosión combinatoria hace que el número de puertas sea mucho mayor que cualquier polinomio.

En cambio, si el circuito tiene tamaño polinómico, tendrá también anchura polinómica, por lo que habrá un nivel en el que no puedan cubrirse todas las combinaciones de salidas del nivel anterior. Las puertas siguientes estarán en la obligación de compartir algún pariente entre sí. Este fenómeno aumentará, por defecto, la cantidad de endogamia del circuito.

Los parientes compartidos más veces se volverán más predominantes a lo largo del circuito. Esto creará una asimetría entre algunas combinaciones de términos y otras: necesariamente algunas combinaciones de entradas estarán sobrerrepresentadas en el circuito en comparación con otras. Los circuitos polinómicos son necesariamente asimétricos en el trato que dan a las entradas y sus combinaciones. Entonces, ¿son necesariamente asimétricas las funciones booleanas que computan?

Vamos a probar que, en contraposición de lo ya comentado, las funciones más simétricas —aquellas que son invariantes ante permutaciones de sus entradas —tienen circuitos polinómicos que las computan.

Teorema 3.1.1. Si $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ es invariante ante permutaciones de sus entradas, entonces existe un circuito de tamaño polinómico con respecto a n que la computa.

Demostración. Sea $x \in \mathbb{Z}_2^n$ una posible entrada de f. Entonces $y \in \mathbb{Z}_2^n$ es una permutación de x si y solo si ambos tienen la misma cantidad de coordenadas con valor 1. Y, en ese caso, se cumple que $f(x) = f(y)$.

Esto quiere decir que, para definir completamente la función f, no hace falta fijar 2^n valores, sino solamente $n + 1$. En concreto, bastará con fijar los valores de $f(\text{bin}(2^i - 1))$ para $0 \leq i \leq n$, pues cualquier otra entrada tendrá la misma cantidad de valores a 1 que alguno de ellos: desde 0 hasta n valores. Podemos representar cada elemento de este conjunto de $n + 1$ números mediante $\lceil \log(n + 1) \rceil$ bits.

El problema de contar cuántos valores a 1 hay en una secuencia de bits está en la clase P. Por tanto, está en $\mathsf{P}_{/\mathsf{poly}}$, así que existen $\lceil \log(n + 1) \rceil$ circuitos de tamaño polinómico que computan cada uno de los $\lceil \log(n+1) \rceil$ bits que conforman ese resultado.

Podemos considerar la función auxiliar $\tilde{f} : \mathbb{Z}_2^{\lceil \log(n+1) \rceil} \to \mathbb{Z}_2$, que cumple $\tilde{f}(x) = f(\text{bin}(2^x - 1))$. El circuito que calcula la Fórmula Normal Disyuntiva de una función con m entradas, expandiendo cada cláusula una a una, es del orden de $O(2^m)$ puertas. En concreto para $\tilde{f}$, existe un circuito que lo computa de tamaño $O(2^{\lceil \log(n+1) \rceil}) = O(2(n + 1)) = O(n)$, que es polinómico en n.

Alimentando este circuito polinómico con las salidas de los $\lceil \log(n+1) \rceil$ circuitos polinómicos descritos previamente, se obtiene un circuito polinómico que calcula la función f. Un esquema para construirlo se encuentra en la figura 3.1. □

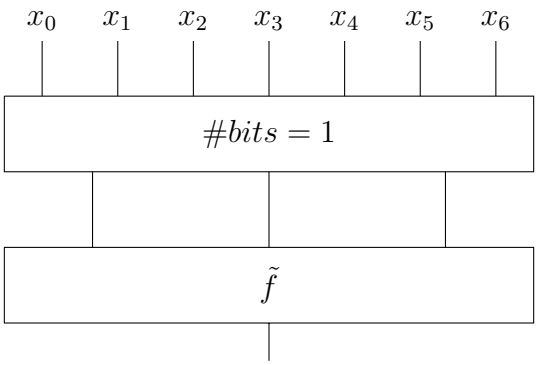

Figura 3.1: Circuito de tamaño polinómico que computa f. Para computar $\tilde{f}$ puede usarse un multiplexor de $n+1$ entradas constantes: $f(\text{bin}(2^i - 1))$ con $0 \leq i \leq n$.

Corolario 3.1.1.1. No solo el circuito construido es polinómico con respecto al tamaño de entrada, sino que también es lineal.

Como ya hemos indicado, el cálculo de $\tilde{f}$ puede realizarse en $O(n)$ puertas. Por otro lado, el contador de bits a 1 puede implementarse a partir de un sumador completo de $5n$ puertas. Por tanto, nuestra construcción requiere, como máximo, $5n + O(n)$ puertas. En resultados recientes, la cota superior de puertas para cualquier función invariante se ha conseguido rebajar a $4.5n + o(n)$ [DKKY10].

No obstante, pensamos que deben existir otras manifestaciones más sutiles de equidad entre variables que sí obliguen a los circuitos a evaluar tantas combinaciones de entradas como para llegar a una cota superpolinómica de puertas.

Una de estas posibles definiciones de equidad no trivial es el isomorfismo de grafos. Creemos que cualquier problema que requiera calcular isomorfismos de grafos caerá en estas consideraciones. Si un problema necesita tratar de la misma manera entradas con alguna relación de isomorfismo entre sí, entonces necesita ser invariante bajo este tipo de equidad. Por ejemplo, el problema del CLIQUE es un probable candidato para ello.

Definición 3.1.1. Dado un grafo no dirigido de v vértices y un número natural positivo $k \leq v$, el problema del CLIQUE consiste en determinar si existe un subgrafo con exactamente k vértices que sea cliqué —es decir, que cada vértice del subgrafo sea adyacente a todos los demás.

Observación 8. El problema del CLIQUE es NP-completo e invariante bajo isomorfismo de los grafos que recibe como entrada. Como hemos comentado en el capítulo 1, los problemas NP-completos son los problemas más difíciles de la clase NP, así que son los mejores candidatos para separar NP de la clase P.

3.2. Relación entre endogamia y repetitividad

En el capítulo 2 hemos relacionado el concepto de repetitividad con los diagramas de decisión binarios; en especial, hemos averiguado que las métricas de desorden derivadas de ellos deberían ser invariantes ante permutaciones de sus entradas.

Creemos que la presencia de una alta endogamia en un circuito —además de provocar asimetría en el trato que se da a sus variables y sus combinaciones, como ya hemos visto— también afectará al nivel de repetitividad en las funciones que este pueda computar.

Por ejemplo, la métrica *Cube* mide el número de hojas de un diagrama de decisión binario que, en cada nodo, aplica una conjunción con la variable que más le conviene. Cada hoja define una cláusula conjuntiva de un subconjunto de variables distinto, definido por las variables, negadas o sin negar, escogidas por la rama que conduce a la hoja. Las hojas forman, por tanto, una partición de la tabla de verdad donde el valor de la función se mantiene constante en cada parte, en cada hoja.

Al unir dos funciones f_1 y f_2 con una operación binaria —como una NAND— la función f resultante tenderá a ser más compleja que sus parientes. Por tanto, esperamos que su métrica de desorden —su ausencia de repetitividad— aumente.

Al combinar P_1 y P_2, las dos particiones de las tablas de verdad de f_1 y f_2 creadas por *Cube* en las hojas de cada diagrama, se tendrá en el caso peor una ortogonalidad total entre las cláusulas, donde ninguna podrá simplificarse con la partición contraria. En ese caso, la partición inducida por el diagrama de f surgiría del producto cartesiano entre las particiones, obteniéndose que $Cube(f) = |P_1||P_2|$.

Una simplificación ocurrirá cuando, al mezclar los dos diagramas, se cree una conjunción que contenga una variable y su valor negado; es decir, cuando para cierta x_i combinemos una rama de un diagrama que contiene x_i con una rama del otro diagrama que contiene $\overline{x_i}$. Cuanto más ocurra, menos podrá crecer la métrica de desorden de f. ¿En qué casos es más probable que se dé esta circunstancia?

Consideramos que la respuesta a esta pregunta es que los diagramas de f_1 y f_2 hagan las **mismas preguntas a niveles parecidos**.

Si el diagrama de f_1 tiene a x_i como una de las primeras variables de decisión del árbol, significa que dicha decisión es importante en una gran cantidad de ramas. Notemos que, para ahorrar nodos, las preguntas por las variables que aparezcan en más ramas deberán hacerse más arriba del árbol. Supongamos que, además, el diagrama de f_2 tiene a x_i también en un nodo muy alto. El diagrama de decisión de f se deshará de todas las subramas contradictorias que se forman al combinar ambos. Y, para la variable x_i, esto ocurrirá en un nivel muy alto, provocando que muchas ramas combinadas desaparezcan al contener x_i y $\overline{x_i}$.

Es más, si ambos diagramas preguntan por las mismas variables en sus niveles más altos, el número de ramas combinadas que no contengan *ninguna* contradicción para ninguna de dichas variables —condición necesaria para no ser contradictorias— podrá llegar a ser despreciable.

Vamos a cuantificar el parecido entre niveles de preguntas de dos diagramas.

Definición 3.2.1. Sean $f_1, f_2 : \mathbb{Z}_2^n \to \mathbb{Z}_2$. Sean A_1 y A_2 los diagramas de decisión binarios creados al calcular la métrica *Cube* sobre f_1 y f_2. El **índice de disimilitud de preguntas** entre f_1 y f_2 es

$$\mathcal{I}(f_1, f_2) = \sum_{i=0}^{n-1} |\mathcal{L}(f_1, i) - \mathcal{L}(f_2, i)|$$

donde $\mathcal{L}(f_k, i)$ es la media de altura a la que se encuentra la variable i en el diagrama A_k. Más formalmente, sea Ξ el conjunto de apariciones de la decisión i:

$$\Xi = \{(v_i, a) : \text{el nodo } v_i \in A_k \text{ tiene como decisión } i \text{ y aparece a altura } a\}$$

La media de altura de i, que pondera exponencialmente los nodos más cercanos a la raíz, es 0 si $\Xi = \emptyset$, y si no, es

$$\mathcal{L}(f_k, i) = \frac{1}{|\Xi|} \sum_{(v,a) \in \Xi} 2^{n-a}$$

Ejemplo 6. Sea el tamaño de entrada $n = 3$ y sean las funciones f_1 y f_2 cuyas cadenas son $chain(f_1)$ = '11111010' y $chain(f_2)$ = '11101000'. Los diagramas obtenidos al calcular la métrica *Cube* sobre ellas se encuentran en la figura 3.2.

En la siguiente tabla se encuentran las medias de altura de cada variable $i = 0, 1, 2$ para las dos funciones.

$\mathcal{L}(f_k, i)$	$i = 0$	$i = 1$	$i = 2$
$k = 1$	$2^{3-0} = 8$	0	$2^{3-1} = 4$
$k = 2$	8	$\frac{1}{2}(4 + 4) = 4$	$\frac{1}{2}(2 + 2) = 2$

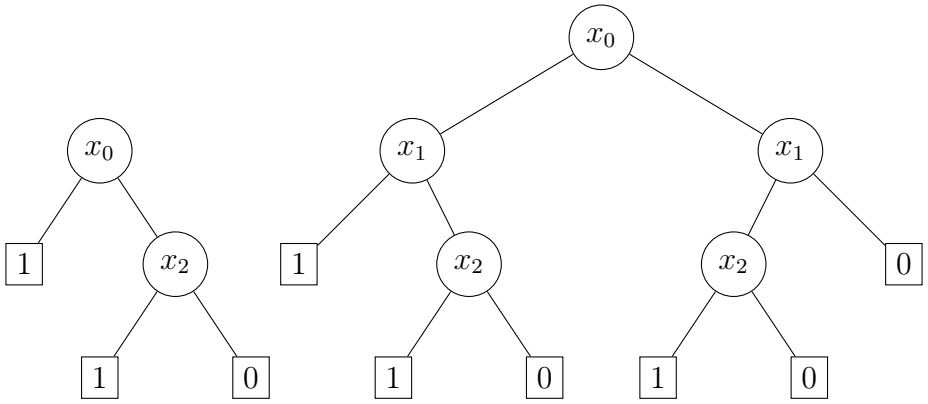

Figura 3.2: Diagramas de decisión de f_1 (izquierda) y f_2 (derecha).

De esta forma, podemos calcular el índice de disimilitud entre ambas funciones como $\mathcal{I}(f_1, f_2) = |8 - 8| + |0 - 4| + |4 - 2| = 6$.

En general, la probabilidad de que, con un índice de disimilitud de preguntas reducido —es decir, que las funciones se parecen en la altura de sus decisiones—, la mezcla entre diagramas no encuentre ningún conflicto entre una variable y su contraria decrecerá rápido con la profundidad de dicho diagrama.

Y, volviendo al comienzo, cuanta más endogamia haya en la mezcla de dos funciones, más bajo esperamos que sea su índice de disimilitud de preguntas, dado que los antecesores comunes compartirán probablemente el orden más conveniente para hacer sus respectivas preguntas sobre variables.

Por este motivo esperamos observar una correlación entre el índice de disimilitud $\mathcal{I}$, la endogamia $\mathcal{E}$ y las métricas de desorden como *Cube*, a la que, por mayor comodidad, denotaremos por $\mathcal{M}$.

3.3. Correlación experimental

Para observar una posible correlación entre las variables mencionadas, debemos disponer de un método experimental que permita analizarlas, tanto entre sí como en función de su crecimiento.

Primero necesitamos generar un conjunto de circuitos aleatorios sobre los que poder medir estos valores. Dado un tamaño $D \in \mathbb{N}$, generaremos circuitos de anchura D y profundidad D, por lo que dispondrán de un máximo de $D \times D$ puertas.

Representaremos cada puerta booleana en un nivel $d \in \{0, \ldots, D-1\}$ como

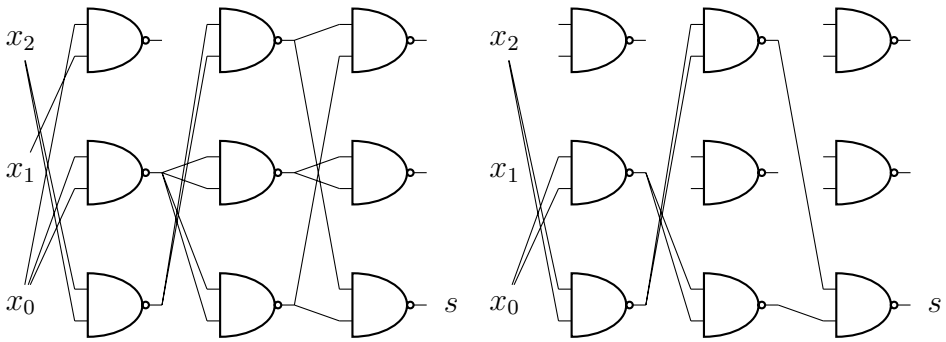

Figura 3.3: Circuito representado por la matriz M_1 del ejemplo 7 (izquierda) y su simplificación sin incluir las conexiones no utilizadas (derecha).

un par de números naturales (p, q), donde p indexa la puerta (o variable) en la posición p del nivel $d-1$ que entra por una de las entradas de dicha puerta; q, equivalentemente, indexa la puerta q del nivel $d-1$ que pasa por la otra entrada. Por tanto $p, q \in \{0, \ldots, D-1\}$.

Observación 9. En el nivel 0, el par (p, q) indexa dos de las entradas directas de la función, ya que no hay un nivel anterior del circuito. Así que $p, q \in \{0, \ldots, n-1\}$.

Definiremos la puerta de salida del circuito como la puerta 0 de la última fila $D-1$, así que la función que computa se calcula a partir de los valores que toma esta puerta.

Ejemplo 7. Generamos aleatoriamente la matriz M_1 de pares con $n = 3$ y $D = 3$, que representa el circuito de la figura 3.3.

$$M_1 = \begin{pmatrix} (2,2) & (0,0) & (0,1) \\ (1,1) & (1,1) & (0,0) \\ (2,0) & (1,1) & (2,0) \end{pmatrix}$$

Muchas de las puertas no acaban conectadas con la salida. Al eliminar las conexiones no productivas, podemos ver que este circuito computa x_0 NAND x_2, que es, precisamente, la función f_1 del ejemplo 6.

Este modelo produce circuitos que, en general, están lejos de tener un tamaño mínimo para la función que computan. Sin embargo, también permite obtener una cota superior fácil de ese tamaño óptimo, que es $D^2 - (D-1)$.

En segundo lugar debemos formalizar qué entendemos exactamente por endogamia. Es una noción sutil a partir de la cual pueden crearse una gran cantidad de variaciones.

En nuestro caso, nos interesa obtenerla a partir de dos circuitos padres que se reúnan en un hijo. Así podremos observar qué ocurre con el crecimiento de las métricas en la generación siguiente, y también podremos aplicar el índice de disimilitud de preguntas, que se obtiene a partir de los dos diagramas de repetitividad padres.

Una primera aproximación a este concepto es contar cuántas puertas comparten entre sí ambos circuitos.

Definición 3.3.1. La **endogamia bilateral** $\mathcal{E}_b(C_1, C_2)$ entre dos circuitos C_1 y C_2 es el número de puertas o entradas que comparten; es decir, de salidas cuya evaluación computa la misma función.

Esta interpretación de la endogamia no tiene en cuenta la cantidad de nodos internamente compartidos de cada circuito por separado. Por tanto, esperamos que lo que mejor refleje sea el crecimiento del desorden con respecto a ambos padres, y no tanto el desorden directo en sí.

El número de puertas puede ponderarse, por ejemplo, en función de a cuántos parientes se corresponde cada puerta, que trataría de mitigar el anterior problema. También se podría ponderar en función de en qué nivel se comparten dichas puertas. Atendiendo a estas modificaciones definimos la segunda noción de endogamia.

Definición 3.3.2. La **endogamia de nivel** $\mathcal{E}_n(C_1, C_2)$ entre dos circuitos C_1 y C_2 de igual profundidad es:

- 1 si son estructuralmente iguales. Si C_1 y C_2 son simplemente entradas, significa que son la misma. Si no lo son, significa que, o bien (C_1^i, C_2^i) y (C_1^d, C_2^d) son dos parejas de subcircuitos estructuralmente iguales, o bien lo son (C_1^i, C_2^d) y (C_1^d, C_2^i).
- 0 si son simplemente entradas y son distintas.
- El máximo entre
$$\frac{1}{2}\mathcal{E}_n(C_1^i, C_2^i) + \frac{1}{2}\mathcal{E}_n(C_1^d, C_2^d)$$
$$\frac{1}{2}\mathcal{E}_n(C_1^i, C_2^d) + \frac{1}{2}\mathcal{E}_n(C_1^d, C_2^i)$$
si no lo son, y donde C^i es el subcircuito padre izquierdo de la última puerta de C, y C^d el derecho.

Ejemplo 8. Tomamos la matriz M_1 del ejemplo 7 y generamos otra del mismo

tamaño a la que llamamos M_2.

$$M_1 = \begin{pmatrix} (2,2) & (0,0) & (0,1) \\ (1,1) & (1,1) & (0,0) \\ (2,0) & (1,1) & (2,0) \end{pmatrix} \qquad M_2 = \begin{pmatrix} (1,2) & (1,1) & (2,0) \\ (2,1) & (2,1) & (0,0) \\ (0,1) & (0,0) & (2,1) \end{pmatrix}$$

Primero eliminamos las puertas no utilizadas.

$$M_1' = \begin{pmatrix} (2,2) & (0,0) & \\ (1,1) & & (0,0) \\ (2,0) & & \end{pmatrix} \qquad M_2' = \begin{pmatrix} & (1,1) & (2,0) \\ (2,1) & (2,1) & \\ (0,1) & & \end{pmatrix}$$

Después calculamos qué funciones computa cada una de las puertas sí utilizadas. Como solo operamos con puertas NAND, denotamos x_i NAND x_j por $\overline{x_i x_j}$ si son distintas, y $\overline{x_i}$ si son iguales.

$$eval(M_1') = \begin{pmatrix} \overline{x_2} & \overline{x_0} & \\ x_0 & & x_2 \\ \overline{x_0 x_2} & & \end{pmatrix} \qquad eval(M_2') = \begin{pmatrix} & \overline{x_1} & \overline{x_0 x_2} \\ \overline{\overline{x_1}\ \overline{x_0 x_2}} & \overline{\overline{x_1}\ \overline{x_0 x_2}} & \\ \overline{x_1}\ \overline{x_0 x_2} & & \end{pmatrix}$$

Podemos ver que comparten la función $\overline{x_0 x_2}$, también comparten las dos entradas x_0 y x_2. Por tanto, la endogamia bilateral $\mathcal{E}_b$ entre ambas es de 3.

Por otro lado, en la fila 0 de cada circuito vemos que ninguna de las puertas es estructuralmente igual, pues todas tienen como entrada pares de variables distintos. En cuanto a las entradas propiamente dichas, el circuito M_1 tiene cuatro veces a 0 y otras cuatro a 2 como parientes —en este caso, bisabuelos, viendo el circuito como un árbol genealógico. El circuito M_2 tiene dos veces a 0, cuatro veces a 1 y dos veces a 2. Así, comparten a cuatro de sus parientes en ese nivel. Por tanto, la endogamia de nivel $\mathcal{E}_n$ entre ambas es de $4 \cdot 2^{0-3} = 0.5$.

Si, por ejemplo, la tercera puerta del primer nivel de M_2 fuera $(0,0)$, ambos circuitos compartirían dos abuelos. Así se tendría que $\mathcal{E}_n = 2 \cdot 2^{1-3} = 0.5$ de nuevo.

La última noción que necesitamos definir es el crecimiento de la métrica de desorden, que tendremos que calcular a partir de las métricas de desorden sobre las dos funciones padre y la función hijo.

Definición 3.3.3. Dadas dos funciones $f_1, f_2 : \mathbb{Z}_2^n \to \mathbb{Z}_2$, sea $f = f_1$ NAND f_2. Si sus valores respectivos de desorden son m_1, m_2 y m, el **crecimiento del desorden** de f con respecto a f_1 y f_2 es $\mathcal{DM} = m - \frac{1}{2}(m_1 + m_2)$.

Para cada valor de $D = \{3, 5, \ldots, 29, 31\}$, hemos generado 10.000 parejas de circuitos de $n = 5$ entradas a los que hemos aplicado $\mathcal{E}_b$, $\mathcal{E}_n$, I y $\mathcal{DM}$. El código

se encuentra en la carpeta C++ del repositorio de este trabajo [RM22b]. A partir de $D = 32$ la precisión de los números flotantes alcanzados disminuye y los valores obtenidos dejan de reflejar la magnitud real.

A continuación hemos generado su matriz de correlación para estimar cuánto se relacionan entre sí estas variables dos a dos. Hemos aplicado el coeficiente de correlación de Spearman, pues calcula la correlación a partir de la posición de cada variable con respecto al resto [Spe87]. Esto permite captar relaciones de monotonía más generales, y no solo lineales como lo hace el coeficiente de Pearson.

Estos son los resultados que hemos obtenido:

1. Endogamia y crecimiento de métrica de desorden.

   Para los pares de circuitos generados, enfrentaremos la endogamia existente entre ambos circuitos con el crecimiento de desorden que se observa en el circuito hijo de ambos —es decir, el que resulta de combinarlos con NAND— con respecto al desorden de sus dos padres.

   Los resultados experimentales muestran que ambas nociones de endogamia definidas tienen una correlación pequeña pero clara, como observamos en la gráfica 3.4. Tienen una relación inversa —es decir, negativa— con $\mathcal{DM}$ porque cuando la endogamia entre los circuitos a combinar aumenta, el crecimiento de desorden en el circuito combinado es menor, tal y como esperábamos.

   Aunque $\mathcal{E}_n$ parece correlacionar mejor para tamaños de circuito pequeños, la relación disminuye hasta llegar a un valor de -0.093. En cambio, $\mathcal{E}_b$ aumenta su correlación lentamente hasta llegar a -0.168.

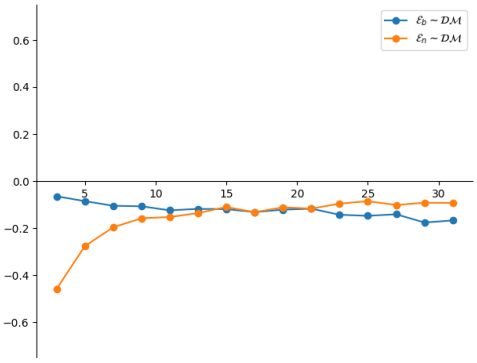

Figura 3.4: Correlación entre $\mathcal{E}_b$ y $\mathcal{E}_n$ con $\mathcal{DM}$ a medida que crece D.

2. Correlación de las endogamias y el desorden diferencial con el índice de disimilitud de alturas.

De muy similar forma ocurre con la correlación de ambas endogamias con el índice $\mathcal{I}$, que en concreto es inversa, como esperábamos. Para $\mathcal{E}_n$ se reduce hasta aparejarse con el coeficiente de $\mathcal{E}_b$ en $D = 19$. Para $\mathcal{E}_b$ el coeficiente mejora ligeramente hasta un valor de -0.175. Podemos verlo en la figura 3.5 a la izquierda.

En cuanto a $\mathcal{I}$ y $\mathcal{DM}$, mantienen una correlación directa alta, pues al aumentar la métrica de desorden también aumenta la distancia entre alturas de los diagramas padres, de nuevo conforme a nuestras expectativas. Con $D = 31$ llega a un valor de 0.538. Esta relación aparece en la figura 3.5 a la derecha.

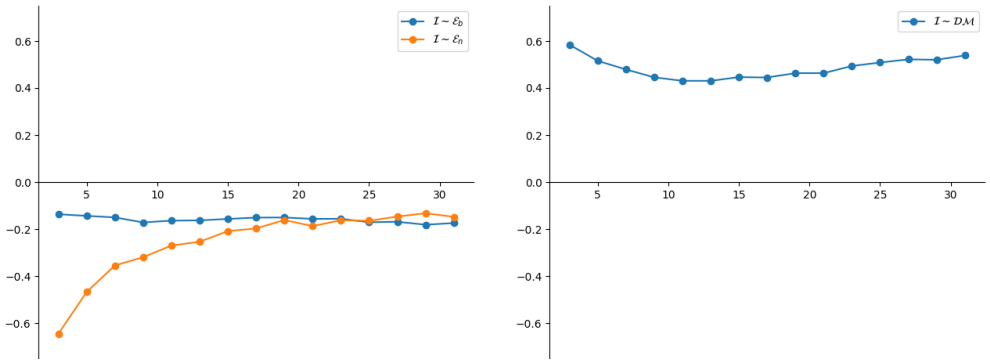

Figura 3.5: Correlación entre $\mathcal{I}$ con $\mathcal{E}_b$ y $\mathcal{E}_n$ (izquierda) y con $\mathcal{DM}$ (derecha) a medida que crece D.

Aumentando el tamaño de los circuitos generados, así como mejorando su generación para evitar situaciones extremadamente redundantes, podríamos ampliar el dominio de estos experimentos y observar si las tendencias encontradas se mantienen para valores de D mayores. Por otro lado, habría que seguir estudiando distintas variaciones de la definición de endogamia. Por ejemplo, se podrían desarrollar nociones que pudieran calcularse para el circuito hijo directamente.

En cualquier caso, existe una ligera correlación inversa palpable entre nuestras definiciones iniciales de endogamia y el crecimiento de la métrica de repetitividad *Cube*. Esto, junto con los experimentos sobre *Cube* realizados en el trabajo de fin de grado en Informática [RM22a], nos hace suponer que ambos conceptos están vinculados al crecimiento del número de puertas de los circuitos booleanos.

Para encontrar el vínculo teórico entre ellas, hemos propuesto el índice de disimilitud de preguntas sobre el diagrama de repetitividad. Hemos visto una alta relación entre dicho índice y la métrica de desorden, que probablemente se beneficie de que ambos valores se calculan sobre el mismo diagrama. Aún más importante

es que hemos encontrado un paralelismo entre el comportamiento de la endogamia con respecto al crecimiento de métrica de desorden y a este índice de disimilitud.

Esto podría justificar que fuera el concepto puente entre ambas, endogamia y desorden. Por un lado, los diagramas *Cube* de circuitos endogámicos tienden a preguntar por las variables en niveles más parecidos. Por otro, una alta coincidencia de variables en los niveles altos de los dos diagramas provoca que, al combinar las ramas de uno con las del otro, muchas se anulan porque contienen al menos un par de variables contradictorias, lo que reduce el desorden del circuito hijo. Esto nos da un modelo, de momento aproximado, de la mecánica de simplificación que ocasiona la endogamia sobre las funciones computadas.

Estas observaciones sobre el significado de la endogamia posiblemente serán muy útiles para intentar generalizar y extender la técnica demostrativa que presentamos en el capítulo 4. Como veremos, esta técnica se basará en forzar un cierto trato exhaustivo en todas las combinaciones de bits de cierto tipo. La endogamia deberá ser clave para poder razonar sobre combinaciones más complejas al obligar al circuito a darles un trato desigual.

Capítulo 4

Cotas inferiores para el tamaño de circuitos booleanos

En este capítulo trataremos de encontrar familias de funciones booleanas tales que, haciéndolas cumplir una serie de requisitos, acaben por necesitar circuitos de anchura determinada para poder ser computadas.

Consideramos que, mediante la generalización de las técnicas propuestas en estos apartados, idealmente podría llegar a encontrarse en el futuro una familia de funciones que requirieran circuitos de gran tamaño con respecto a la entrada, e idealmente superpolinómico.

Concretamente, las limitaciones en anchura aquí propuestas podrían generalizarse a una mayor limitación en la cantidad de puertas total del circuito, y pensamos imprescindible para ello considerar la necesidad de endogamia de cualquier circuito polinómico.

Con técnicas como las que presentamos se podría mostrar que ciertas funciones requieren, en cualquier circuito que las compute, una cantidad alta de subcircuitos diferentes *siempre y cuando* tengan la obligación lidiar con las condiciones impuestas desde el primer nivel del circuito, no pudiendo retrasar tal tarea a un nivel posterior que reaproveche lo ya computado. Por culpa de la endogamia, puede que dejar ese procesamiento para niveles posteriores —y así reaprovechar cómputos anteriores y reducir el número de subcircuitos diferentes necesarios— no sea posible.

Las condiciones impuestas a las funciones consistirán en dar cierto trato uniforme a ciertas combinaciones de bits —pares en los resultados que presentaremos, y mayores en una posible generalización. Sin embargo, la endogamia obligará a

dar un trato privilegiado a algunas combinaciones de bits concretas debido a la repetición de parientes, lo que debería impedir la uniformidad deseada.

En todo caso, las demostraciones de este capítulo son una muestra de la nueva metodología y la base teórica para que, en un futuro, se pudiera desarrollar y alcanzar este resultado.

Para las sucesivas secciones, el conjunto de puertas booleanas de los circuitos será $\{\text{AND}, \text{OR}, \text{NOT}\}$.

4.1. Circuitos en Forma AC

Para poder probar resultados relacionados con la forma concreta de los circuitos booleanos, necesitamos transformarlos a una forma que podamos manejar más cómodamente. A esta forma la hemos llamado *Alternating Circuit Form*, o Forma de Circuito Alternado.

Definición 4.1.1. Un circuito tiene **Forma de Circuito Alternado** (*AC Form*) si:

- Tiene una profundidad, o número de filas de puertas, impar.
- Cada fila toma sus entradas únicamente de la fila anterior.
- Su fila 0 está formada únicamente por puertas NOT o la identidad.
- Las filas impares están formadas por puertas AND.
- El resto de filas pares están formadas por puertas OR.

Observación 10. Las puertas que no sean necesarias en una fila pueden computar la función identidad, como ocurre con la puerta OR de la fila 4 del ejemplo que aparece en la figura 4.1.

Observación 11. Esta forma estándar proviene de la aplicación del Switching Lemma de Håstad [Has86] para probar que el problema Paridad no pertenece a la clase AC^0; es decir, que no puede computarse por una familia de circuitos polinómica con profundidad constante y número de entradas de cada puerta (*fan-in*) ilimitado. Nosotros lo aplicaremos a circuitos generales y con *fan-in* de 2. En adelante solo nos referiremos a circuitos de este tipo.

A continuación mostraremos cómo transformar cualquier circuito en Forma AC, y calcularemos que su tamaño solo crece, a lo sumo, polinómicamente con respecto a su tamaño original.

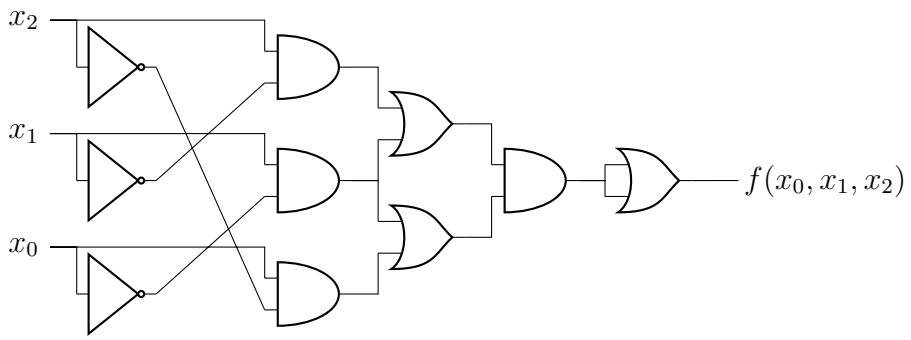

Figura 4.1: Circuito en Forma de Circuito Alternado (AC) que computa la función $f(x_0, x_1, x_2) = (x_1\overline{x_0} + x_2\overline{x_1})(x_1\overline{x_0} + x_0\overline{x_2}) = x_1\overline{x_0}$.

Lema 4.1.1. [Dun92] Si una función $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ puede computarse por un circuito C de tamaño $S(n)$, también puede computarse por un circuito C' con todas sus puertas NOT en la primera fila y de tamaño menor o igual que $2S(n)+n$.

Demostración. Sea una puerta g tal que alguna de sus salidas está conectada a una puerta NOT, que llamaremos h, y tal que, desde h en adelante, no haya ninguna otra puerta NOT. Si g no existe, entonces todas las puertas NOT están en la primera fila.

Si g es otra puerta NOT, h puede eliminarse, y conectar sus salidas a la entrada de g, ya que $\overline{\overline{g_1}} = g_1$ para todo valor de g_1. Podemos ver esta transformación en la figura 4.2.

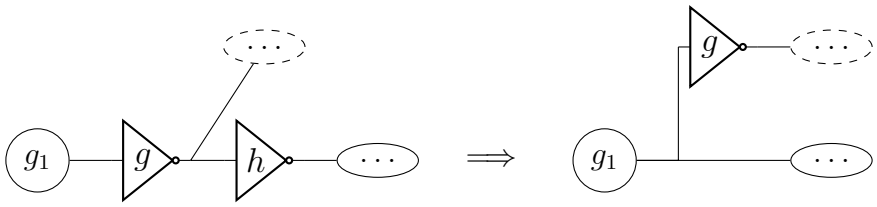

Figura 4.2: Transformación de circuito usando $\overline{\overline{g_1}} = g_1$.

Si g no es una puerta NOT, sean g_1 y g_2 sus entradas. Si hay otra puerta NOT $\hat{h}$ conectada a la salida de g, está calculando el mismo valor que h, luego puede sustituirse por h. Podemos ver esta transformación en la figura 4.3.

A continuación, aplicamos las leyes de De Morgan al par (g, h). Si g es una puerta OR, se añade una nueva puerta AND g' con dos puertas NOT en sus entradas, ya que $\overline{g_1 + g_2} = \overline{g_1}\,\overline{g_2}$. Si g es una puerta AND, entonces g' será una OR, ya que $\overline{g_1 g_2} = \overline{g_1} + \overline{g_2}$. Podemos ver esta transformación en la figura 4.4.

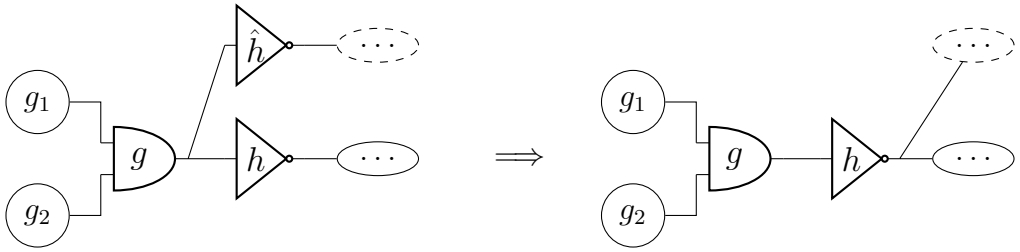

Figura 4.3: Transformación de circuito mediante la eliminación de puertas NOT innecesarias.

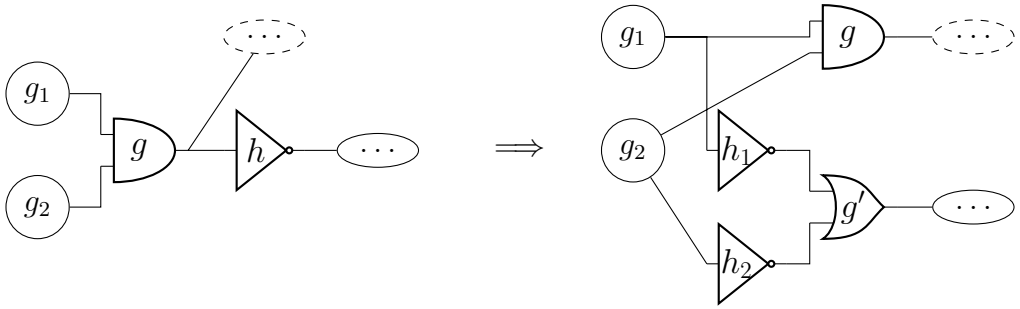

Figura 4.4: Transformación de circuito usando $\overline{g_1 g_2} = \overline{g_1} + \overline{g_2}$.

Cuando no queden más puertas g con puertas NOT detrás, todas las puertas NOT estarán en la primera fila del circuito, como queríamos conseguir. Por cada puerta AND u OR del circuito hemos añadido, a lo sumo, una única puerta más. Además, el número máximo de puertas NOT en la primera fila es el de las entradas, n. Por tanto, C' tendrá un tamaño máximo de $2S(n) + n$. □

Observación 12. Una prueba alternativa y más directa consiste en tomar el circuito original sin puertas NOT, duplicarlo, cambiar todas sus puertas AND por OR y viceversa, y poner puertas NOT en todas las entradas del circuito duplicado. Cuando se llega a un punto en el circuito en el que habría habido una puerta NOT, se toma el valor del circuito duplicado, que, por las leyes de De Morgan, tendrá el valor negado que se busca.

Observación 13. Los circuitos con puertas AND, OR y NOT cuyas puertas NOT se encuentran en la primera fila son denominados de diversas formas: circuitos DeMorgan, circuitos normalizados (*normalized*), circuitos estándar (*standard*) o circuitos con negaciones ajustadas (*with tight negations*) [JL22].

Lema 4.1.2. Si una función $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ puede computarse por un circuito C monótono —sin puertas NOT— de profundidad $d(n)$ y tamaño $S(n)$, también puede computarse por un circuito C' cuyas filas alternan puertas AND y OR, de profun-

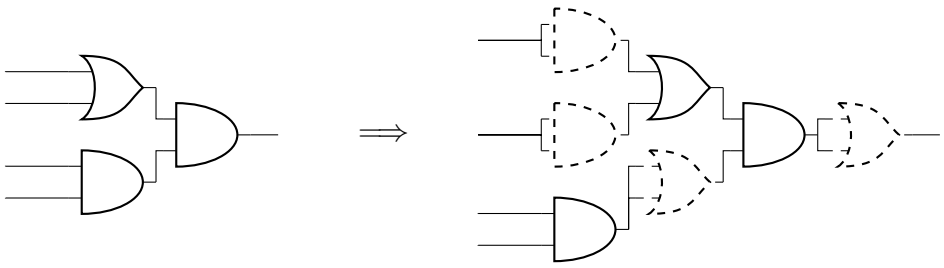

Figura 4.5: Transformación descrita en la prueba del lema 4.1.2.

didad menor o igual que $2d(n)$ y tamaño menor o igual que $S(n)+d(n)(S(n)+n)$.

Demostración. Sea $\mathcal{V}_i$ el conjunto de puertas tales que se encuentran a i puertas de la puerta de salida, con $0 \leq i \leq d(n)$. Numerando las filas de la salida hacia las entradas, para cada i se disponen en la fila $2i$ las puertas OR de $\mathcal{V}_i$, y en la fila $2i-1$ las puertas AND de $\mathcal{V}_i$.

Por último, si existe una salida g y una puerta h tal que el número de filas entre ambas es $k > 1$, se añadirá una puerta extra por cada fila intermedia. Estas puertas extra computarán la función identidad, y tomarán por sus dos entradas la salida de la fila anterior, que tendrá el mismo valor que la salida en g.

En concreto, si la puerta de salida es una AND, se añade una puerta OR justo detrás, y toma como entradas la salida de esa AND. Así, C' tiene una profundidad, como máximo, de $2d(n)$. Podemos ver esta transformación en la figura 4.5.

El número de posibles salidas g es, como máximo, $S(n)+n$ —el número de puertas más el de entradas—, y el número de puertas extra a añadir por cada elección de g es de, como máximo, $d(n)$. Por tanto, el circuito C' tendrá un tamaño de, a lo sumo, $S(n)+d(n)(S(n)+n)$. □

Teorema 4.1.3. Si una función $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ puede computarse por un circuito C de profundidad $d(n)$ y tamaño $S(n)$, también puede computarse por un circuito C' en Forma AC de profundidad menor o igual que $2d(n)$ y tamaño menor o igual que $2S(n)+n+2d(n)(S(n)+n)$.

Demostración. Aplicando el lema 4.1.1 a C se obtiene un circuito C''. El circuito C'' puede verse como un circuito con $2S(n)$ puertas AND y OR, profundidad $d(n)$, y $2n$ entradas donde una mitad son las otras n negadas. Aplicando a C'' el lema 4.1.2, se obtiene un circuito C' en Forma AC que cumple las cotas resultantes. □

Observación 14. Lo que verdaderamente nos resultará útil del resultado del teorema 4.1.3 es que la transformación entre un circuito general y uno en Forma AC es polinómica.

4.2. Definición y aplicación de la técnica

Nos gustaría encontrar alguna función concreta que, por construcción, sea suficientemente compleja como para que cualquier circuito en Forma AC que lo compute necesite una gran cantidad de puertas.

Vamos a presentar una técnica demostrativa que podría utilizarse para conseguir dicho fin. En concreto, usaremos las características de la Forma AC para restringir iterativamente una función genérica f, de manera que requiera una anchura determinada en algún nivel de cualquier circuito que la compute.

En primer lugar, debemos definir qué significa que una función dependa de sus entradas, o, dicho con otras palabras, que sus entradas importen a la hora de computar la función.

Definición 4.2.1. La función $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ **depende** de su variable i si existe un par de vectores $v_i^0, v_i^1 \in \mathbb{Z}_2^n$, que son iguales salvo en la coordenada i —$(v_i^0)_i = 0$, $(v_i^1)_i = 1$ y $(v_i^0)_j = (v_i^1)_j \ \forall j \neq i$—, para las que $f(v_i^0) \neq f(v_i^1)$.

En general, si $f(v_i^1) = 1$ diremos que es x_i quien **importa** a f, y si $f(v_i^0) = 1$ diremos que es $\overline{x_i}$ quien importa a f. Además, al par (v_i^0, v_i^1) lo denominamos como un par de **vectores contexto** de la variable x_i (o $\overline{x_i}$).

Teorema 4.2.1. Sea $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ tal que depende de x_i, y sean $v_i^0, v_i^1 \in \mathbb{Z}_2^n$ vectores contexto de x_i. Sea además C un circuito en Forma AC que computa f. Existe un camino en el circuito desde la variable x_i hasta la salida que al evaluar $f(v_i^1)$ sobre C vale constantemente 1, y que al evaluar $f(v_i^0)$ vale constantemente 0.

Demostración. Consideramos la evaluación de $f(v_i^1)$ sobre C. Como la puerta de salida —una OR— es un 1, al menos una de sus dos entradas es un 1. Este 1 es la salida de una AND del penúltimo nivel, cuyas dos entradas deben valer también 1. Así, consideramos el grafo $\mathcal{A}$ formado por todas las puertas y todas las conexiones que valen constantemente 1 desde la salida, que, repitiendo el argumento previo en todos los niveles, debe llegar al primer nivel para alguna de sus ramas.

Ahora consideramos la evaluación de $f(v_i^0)$ sobre C, que devuelve un 0. Esto es porque la última puerta —una OR— tiene como entradas dos 0; en concreto, lo tiene la entrada que antes valía 1 y que pertenece a $\mathcal{A}$. Ese 0 es la salida de la AND

considerada antes, y que ahora tiene alguna de sus dos entradas a 0. Repitiendo el argumento de nuevo por ese camino, se da que existe un camino en $\mathcal{A}$ que vale constantemente 0 y que llega al primer nivel.

El último valor del camino conecta con alguna de las entradas, negadas o sin negar. Pero tenemos que, para v_i^1, vale 1, y para v_i^0 vale 0. La única variable que cumple esto es x_i. □

Observación 15. Si un circuito general C computa una función f para la que todas sus entradas importan, C tiene una anchura mínima de $\frac{n}{2}$ puertas en su primer nivel, pues sus n entradas pueden emparejarse y entrar por $\frac{n}{2}$ puertas AND en su primer nivel. Y, de hecho, no pueden entrar todas con una anchura menor.

Si toda variable x_i importa y toda variable negada $\overline{x_i}$ importa, las $2n$ variables deberán pasar por, al menos, alguna puerta del primer nivel, y la forma más reducida posible es emparejándolas entre sí, lo que requiere n puertas en ese nivel.

Vamos a mostrar un ejemplo de cómo aplicar nuestra técnica demostrativa para obtener esta cota tan sencilla. La técnica consiste en construir una familia de funciones a través de la valoración de sus salidas, de tal manera que el circuito que las compute requiera una anchura mínima. En este caso, conseguimos que f solo dependa de n de las $2n$ posibles entradas, reduciendo el requisito del argumento anterior.

Teorema 4.2.2. Existe una familia de funciones booleanas $\mathcal{F}_1$ (dependiente de n) tales que, si $f \in \mathcal{F}_1$, cualquier circuito en Forma AC que la compute tendrá una anchura de, al menos, n puertas en su fila 1.

Demostración. Sean $v_0^1 = (1, 1, \ldots, 1)$ y $v_0^0 = (0, 1, \ldots, 1)$. Fijamos los valores de $f(v_0^1) = 1$ y $f(v_0^0) = 0$ para que la variable x_0 importe.

Para el primer caso, el valor de $x_0 = 1$ tendrá que propagarse a lo largo del circuito, como hemos determinado en el teorema 4.2.1. Así, la entrada x_0 deberá conectarse, al menos, con una de las puertas AND del nivel 1. A esta puerta AND la llamaremos A_0.

Fijamos ahora el valor de $f(0, 0, 1, \ldots, 1) = 1$. Por tanto, $\overline{x_1}$ importa. Así, deberá conectarse con al menos otra de las puertas del nivel 1, A_1. No puede ocurrir que $A_0 = A_1$, ya que, al ser $x_0 = 0$, A_0 siempre devolverá el valor 0 para esa entrada concreta.

De igual manera, fijamos $f(0, 1, \ldots, \overset{i)}{0}, \ldots, 1) = 1$ para el resto de variables $i \neq 0$. Cada $\overline{x_i}$ importa, y pasará por alguna puerta AND del nivel 1, A_i. No puede

ocurrir que $A_0 = A_i$ de nuevo, y tampoco que $A_i = A_j$ para $j \neq 0$, pues, como $x_j = 1$, entonces $\overline{x_j} = 0$ y la puerta A_j devolverá 0.

Como $A_i \neq A_j$ para todo $0 \leq i \neq j < n$, entonces la fila de puertas AND del nivel 1 tiene, al menos, una anchura de n puertas. Podemos observar el resultado en la figura 4.6. □

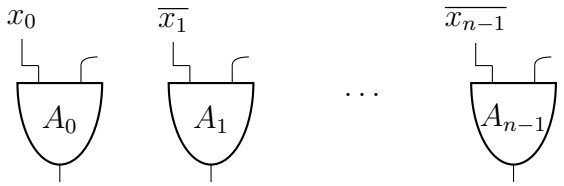

Figura 4.6: Configuración del primer nivel de puertas AND de los circuitos que computan funciones de la familia $\mathcal{F}_1$ para el teorema 4.2.2.

Nos gustaría, sin embargo, obtener un resultado más útil que una cota de n en la anchura del circuito. Vamos a demostrar que existen funciones que requieren, al menos, $2n$ puertas de anchura en su primer nivel.

Teorema 4.2.3. Existe una familia de funciones booleanas $\mathcal{F}_2$ tales que, si $f \in \mathcal{F}_2$, cualquier circuito en Forma AC que lo compute tendrá una anchura de, al menos, $2n$ puertas en su primer nivel.

Demostración. Sea un par de índices distintos $i, j \in \{0, \ldots, n-1\}$. Para todo vector $v_{i,j}^{p,q}$, se toma por notación que $(v_{i,j}^{p,q})_i = p$ y $(v_{i,j}^{p,q})_j = q$. Es decir, los subíndices representan un par de coordenadas (i, j) y los superíndices representan el valor que toma ese par de coordenadas o variables en este vector de entrada, con $p, q \in \mathbb{Z}_2$.

1. Sean unos vectores $a_{i,j}^{0,0}$ y $a_{i,j}^{1,0}$ tales que $(a_{i,j}^{0,0}, a_{i,j}^{1,0})$ son vectores contexto de $\overline{x_i}$.

   Es decir, $f(a_{i,j}^{0,0}) = 1$ y $f(a_{i,j}^{1,0}) = 0$.

2. Sean unos vectores $b_{i,j}^{0,1}$ y $b_{i,j}^{1,1}$ tales que $(b_{i,j}^{0,1}, b_{i,j}^{1,1})$ son vectores contexto de $\overline{x_i}$.

   Es decir, $f(b_{i,j}^{0,1}) = 1$ y $f(b_{i,j}^{1,1}) = 0$.

3. Sean unos vectores $c_{i,j}^{1,0}$ y $c_{i,j}^{0,0}$ tales que $(c_{i,j}^{1,0}, c_{i,j}^{0,0})$ son vectores contexto de x_i.

   Es decir, $f(c_{i,j}^{1,0}) = 1$ y $f(c_{i,j}^{0,0}) = 0$.

4. Sean unos vectores $d_{i,j}^{1,1}$ y $d_{i,j}^{0,1}$ tales que $(d_{i,j}^{1,1}, d_{i,j}^{0,1})$ son vectores contexto de x_i.

   Es decir, $f(d_{i,j}^{1,1}) = 1$ y $f(d_{i,j}^{0,1}) = 0$.

Estos vectores pueden construirse de tal forma que no se contradicen entre sí para cualquier pareja de índices (i, j). Solo hay que asegurar que aquellos que evalúan a 1 sean distintos que los que evalúan a 0; veamos cómo.

Para cada uno de los cuatro casos considerados, el vector que evalúa a 1 y el que evalúa a 0 son iguales salvo un índice cada uno. Por tanto, tienen paridad contraria.

Basta con tomar $a_{i,j}^{0,0}$, $b_{i,j}^{0,1}$, $c_{i,j}^{1,0}$ y $d_{i,j}^{1,1}$ con un número de entradas a 1 impar, sin importar si se repiten entre sí para distintas combinaciones de índices, y fijar que f los evalúe a 1. Después, y por definición, sus contrapartes pueden evaluarse a 0: al tener número de entradas a 1 par, no pueden haber sido definidas en el paso anterior.

Como máximo, hemos definido un total de $4 \cdot 2 \cdot (n^2 - n)$ salidas para $f \in \mathcal{F}_2$. Consideramos ahora un circuito en Forma AC que compute f.

Sea x_i una entrada cualquiera. Hemos asegurado que f dependa de ella, luego pasará por, al menos, una puerta AND A del primer nivel. Supongamos que x_j es la otra entrada de esta puerta. Si $i = j$ y está emparejada consigo misma, ocupa dos de las posibles entradas de A.

Si, en cambio, está emparejada con $\overline{x_i}$, A devuelve siempre el valor constantemente a 0. Como la variable importa, tiene que pasar por alguna otra puerta distinta a A.

Si $i \neq j$, vamos a probar que x_i debe pasar al menos por otra puerta AND del primer nivel diferente de A. Con esto habremos probado que x_i debe pasar al menos por dos entradas en cualquiera de sus configuraciones.

Consideramos el par de contextos $(c_{i,j}^{1,0}, c_{i,j}^{0,0})$, y la evaluación de $f(c_{i,j}^{1,0})$ a 1. Por el teorema 4.2.1, existe un camino desde la variable x_i hasta la salida que es constantemente 1. Sin embargo, la salida de la puerta A es 0, ya que $(c_{i,j}^{1,0})_j = 0$, por lo que ese camino no pasa por A. Esto quiere decir que x_i debe pasar por otra puerta además de A.

Similarmente, si la otra entrada que va con x_i en A es $\overline{x_j}$, consideramos el mismo argumento con el par $(d_{i,j}^{1,1}, d_{i,j}^{0,1})$. De igual forma, para las entradas negadas $\overline{x_i}$, consideramos $(a_{i,j}^{0,0}, a_{i,j}^{1,0})$ y $(b_{i,j}^{0,1}, b_{i,j}^{1,1})$, respectivamente, dependiendo de que se empareje con x_j o $\overline{x_j}$. En definitiva, para las $2n$ entradas —contando sus formatos con y sin negación— cada una debe pasar por, al menos, dos entradas distintas de algunas de las puertas del primer nivel. Esto solo es posible si el primer nivel tiene al menos $2n$ puertas.

Además, si el circuito tiene una anchura en el primer nivel de exactamente $2n$

puertas, cada una de las $2n$ entradas debe pasar exactamente dos veces, pues no pueden pasar menos. □

4.3. Uso futuro de la técnica

De la misma forma que hemos diseñado una serie de entradas de funciones booleanas por parejas para comprometer la anchura del primer nivel de puertas AND, pensamos que puede idearse un sistema similar para restringir las posibilidades del siguiente nivel de puertas OR.

Mientras que en el primer nivel razonamos sobre la supervivencia de los valores a 1 que cruzan las AND, en este nivel son los valores a 0 los que deben encontrar caminos a través de ambas entradas de alguna de las puertas OR. Y, en vez de definir pares de entradas incompatibles entre sí, quizás hagan falta un número más elevado de restricciones, como, por ejemplo, cuartetos de entradas.

En este segundo nivel no buscaríamos justificar una anchura de puertas determinada, sino que nos gustaría averiguar cuántos subcircuitos son necesarios para que sobrevivan los caminos constantemente nulos desde las entradas hasta la salida.

Para algunas de las restricciones haría falta una configuración de pares de variables concreta. Para otras, haría falta la contraria. Afinando las relaciones entre ellas, se trataría de obtener la mayor discordancia posible, que produciría el mayor número de subcircuitos distintos necesarios para satisfacerlo.

Por último, la técnica tendría que llevarse a cabo para el resto de niveles posteriores involucrando combinaciones mayores de las variables de input: de ocho en ocho, dieciséis, etc.

Este proceso podría encontrarse con algunos obstáculos para poder llevarse a cabo. En primer lugar, es necesario que las restricciones sigan siendo consistentes entre sí. También haría falta saber cuánto crece esa cantidad de subcircuitos distintos necesarios en función del número de restricciones.

Asimismo, al razonar sobre los primeros niveles del circuito podemos permitirnos considerar los *inputs* originales. Sin embargo, cuando las explicaciones se generalicen, hará falta razonar sobre salidas de subcircuitos computados parcialmente. Consideramos que el argumento podrá adaptarse para seguir funcionando en el caso asintótico.

Cabe la posibilidad de que, en anchura, el circuito requiera una gran cantidad de subcircuitos, pero que alternativamente pueda aprovechar niveles posteriores para ir computándolos de manera menos costosa, reaprovechando los cómputos

realizados en los niveles anteriores. Lo que proponemos para extender nuestra técnica es basarnos en que la endogamia deberá, previsiblemente, imposibilitar ese aprovechamiento en altura debido a la explosión combinatoria de entradas comprometidas en la construcción: la endogamia necesitará dar un tratamiento asimétrico, privilegiando a unas combinaciones sobre otras, que no satisfará a algunas de las entradas comprometidas.

Idealmente, en algún punto del circuito habremos comprometido un número $m < 2^n$ de entradas de la familia de funciones tales que, en total, se requiriera un número de subcircuitos elevado. Supongamos que fuera, por ejemplo, n^2 o 2^n.

El primer caso probaría que cualquier circuito que computara alguna de estas funciones necesitaría un número cuadrático de puertas con respecto a la entrada. La cota inferior más alta conocida del tamaño de circuitos de carácter general que computan algún problema en NP es lineal [LY22], por lo que una cota cuadrática sería un descubrimiento interesante en cualquier caso, y, en particular, extraordinario en este campo de investigación, si el problema definido por la familia de funciones considerada en la construcción estuviera en NP.

El segundo caso mencionado antes, es decir, que el número de subcircuitos diferentes necesarios fuera una cantidad superpolinómica tal como 2^n, probaría, si el problema construido perteneciera a NP, la conjetura $\mathsf{P} \neq \mathsf{NP}$.

Capítulo 5

Conclusiones

La búsqueda de cotas inferiores del tamaño de los circuitos booleanos es un largo proceso que requiere de una gran cantidad de ideas y pasos a seguir. Hemos estudiado sus antecedentes y mostrado por qué la respuesta a esta pregunta podría llevar a resolver P vs. NP, uno de los Problemas del Milenio.

En el capítulo 4 hemos abordado la cuestión restringiéndonos a un tipo de circuitos concretos, aquellos con Forma de Circuito Alternado, que tienen todas sus puertas NOT en el nivel de las entradas y alternan puertas AND y OR en el resto de niveles. Hemos demostrado que pasar a esta forma cuesta una cantidad polinómica de puertas desde un circuito arbitrario —con esos tres tipos de puertas y *fan-in* igual a dos.

A partir de la estructura de estos circuitos en Forma AC hemos desarrollado una técnica demostrativa para encontrar cotas inferiores de su número de puertas. Se basa en la construcción *ad hoc* de funciones cuyas salidas sean incompatibles con todas las posibles parejas de variables que puedan pasar por cada puerta. A modo de ilustración, hemos aplicado la técnica para encontrar una cota de n puertas en el primer nivel de AND. Posteriormente la hemos mejorado y hemos llegado a una cota inferior de $2n$ puertas en ese mismo nivel.

Consideramos que esta técnica podría utilizarse sobre los siguientes niveles del circuito. A la hora de realizar esta extensión, tenemos que entender más profundamente por qué algunos cómputos pueden retrasarse a niveles posteriores del circuito y beneficiarse de reutilizar cómputos anteriores. Y, sobre todo, entender qué condiciones sobre la función dificultarán tal opción.

Como hemos explicado anteriormente, el concepto de endogamia podría ayudarnos en ese respecto. Consiste en considerar el número de subcircuitos repetidos

o compartidos entre distintas ramas de un circuito. Pero antes de poder utilizarlo en una demostración necesitamos entender su mecánica interna, de la cual nuestros experimentos realizados en el capítulo 3 nos han proporcionado una primera aproximación.

Además, conjeturamos que las funciones que manifiestan algún tipo de equidad compleja entre las variables, como el isomorfismo de grafos, no podrán computarse con circuitos pequeños por culpa de la endogamia, pues estarán obligados a evaluar con cierta equidad un amplio rango de combinaciones de entradas. Sin embargo, también hemos demostrado que para el tipo de equidad más sencillo, la simetría entre variables, las funciones que la presentan sí son polinómicas.

En el trabajo de fin de grado asociado al Grado en Ingeniería Informática [RM22a] se desarrollaron definiciones de repetitividad o desorden de funciones booleanas capaces de clasificar acertadamente un *dataset* de millones de funciones en función del tamaño de sus circuitos mínimos. Al haber obtenido un porcentaje de acierto de más del 95 % utilizando directamente tales métricas, decidimos utilizarlas como estimadores de ese tamaño —el resto de dicho trabajo derivó hacia el uso y estudio de clasificaciones basadas en redes neuronales. En este trabajo hemos demostrado muchas de las afirmaciones sobre las métricas de desorden que en el trabajo asociado se habían quedado sin probar. Por ejemplo, hemos añadido una semántica a la sintaxis que genera a una de ellas.

Las métricas desarrolladas se basan en intuiciones básicas sobre qué significa ser repetitiva para una función booleana. Se han mejorado paulatinamente las definiciones hasta obtener unos diagramas o árboles de decisión no ordenados. A su total de hojas se le llama la métrica $Cube$, pues representa el proceso de división en subcubos de un hipercubo en el espacio n-dimensional, donde n es el número de entradas de la función.

Con esta métrica de desorden como estimador del tamaño mínimo de los circuitos, hemos buscado comprobar si nuestras intuiciones sobre la endogamia son válidas y si se relacionan con el desorden de una función. Para conseguirlo, hemos definido variaciones del concepto de endogamia, como la endogamia bilateral o de nivel, para después calcular la correlación que existe entre ellas y el desorden.

No solo hemos tratado de verificar experimentalmente que la endogamia evita el crecimiento de desorden de los circuitos, sino que también hemos intentado entender por qué. Así, hemos definido un concepto que pretende hacer de puente entre ambas nociones: la distancia o disimilitud entre preguntas del diagrama de decisión. Su correlación con el crecimiento de desorden y con la endogamia podría justificar ese posible papel entre ambos.

De esta forma es como presentamos nuestra propuesta —original hasta donde sabemos— para buscar en el futuro funciones que requieran circuitos superpolinómicos para ser computadas. En el proceso, hemos entendido más profundamente algunos de los motivos y mecanismos por los que esto sucede.

Bibliografía

[AB87] Alon, Noga y Ravi B. Boppana: *The monotone circuit complexity of boolean functions*. Combinatorica, 7(1):1–22, Marzo 1987. https://doi.org/10.1007/BF02579196.

[AB09] Arora, Sanjeev y Boaz Barak: *Boolean circuits*. En *Computational Complexity: A Modern Approach*, capítulo 6, página 106–122. Cambridge University Press, 2009. https://doi.org/10.1017/CBO9780511804090.009.

[BGS75] Baker, Theodore, John Gill y Robert Solovay: *Relativizations of the P=?NP Question*. SIAM Journal on Computing, 4(4):431–442, 1975. https://doi.org/10.1137/0204037.

[Blu83] Blum, Norbert: *A Boolean function requiring 3n network size*. Theoretical Computer Science, 28(3):337–345, 1983. https://doi.org/10.1016/0304-3975(83)90029-4.

[Bry86] Bryant, Randal E.: *Graph-Based Algorithms for Boolean Function Manipulation*. IEEE Transactions on Computers, C-35(8):677–691, 1986. https://doi.org/10.1109/TC.1986.1676819.

[BSSW10] Bollig, Beate, Martin Sauerhoff, Detlef Sieling y Ingo Wegener: *Binary Decision Diagrams*. En Crama, Yves y Peter L. Hammer (editores): *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, Encyclopedia of Mathematics and its Applications, capítulo 10, página 473–505. Cambridge University Press, 2010. https://doi.org/10.1017/CBO9780511780448.013.

[Can91] Cantor, Georg: *Ueber eine elementare Frage der Mannigfaltigketislehre*. Jahresbericht der Deutschen Mathematiker-Vereinigung, páginas 75–78, 1891. https://gdz.sub.uni-goettingen.de/id/PPN37721857X_0001.

[Coo71] Cook, Stephen: *The Complexity of Theorem-Proving Procedures*. En *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, página 151–158. Association for Computing Machinery, 1971. https://doi.org/10.1145/800157.805047.

[Coo00] Cook, Stephen: *The P versus NP problem*. The Millennium Prize Problem, 2000. https://www.claymath.org/sites/default/files/pvsnp.pdf.

[DKKY10] Demenkov, Evgeny, Arist Kojevnikov, Alexander S. Kulikov y Grigory Yaroslavtsev: *New upper bounds on the Boolean circuit complexity of symmetric functions*. Information Processing Letters, 110(7):264–267, 2010, ISSN 0020-0190. https://doi.org/10.1016/j.ipl.2010.01.007.

[Dun92] Dunne, Paul E.: *Relationships Between Monotone and Non-Monotone Network Complexity*. En Paterson, Michael S. (editor): *Boolean Function Complexity*, London Mathematical Society Lecture Note Series, página 1–24. Cambridge University Press, 1992. https://doi.org/10.1017/CBO9780511526633.003.

[FGHK16] Find, Magnus Gausdal, Alexander Golovnev, Edward A. Hirsch y Alexander S. Kulikov: *A Better-Than-3n Lower Bound for the Circuit Complexity of an Explicit Function*. En *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, páginas 89–98, 2016. https://doi.org/10.1109/FOCS.2016.19.

[GHKK18] Golovnev, Alexander, Edward A. Hirsch, Alexander Knop y Alexander S. Kulikov: *On the limits of gate elimination*. Journal of Computer and System Sciences, 96:107–119, 2018, ISSN 0022-0000. https://doi.org/10.1016/j.jcss.2018.04.005.

[Has86] Hastad, Johan: *Almost Optimal Lower Bounds for Small Depth Circuits*. En *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, página 6–20. Association for Computing Machinery, 1986. https://doi.org/10.1145/12130.12132.

[JL22] Jukna, Stasys y Andrzej Lingas: *Lower bounds for Boolean circuits of bounded negation width*. Journal of Computer and System Sciences, 129:90–105, 2022. https://doi.org/10.1016/j.jcss.2022.05.003.

[KL82] Karp, Richard M. y Richard J. Lipton: *Turing machines that take advice*. L'Enseignement mathématique, 28:191–209, 1982. https://doi.org/10.5169/seals-52237.

[Lee59] Lee, C. Y.: *Representation of switching circuits by binary-decision programs*. The Bell System Technical Journal, 38(4):985–999, 1959. https://doi.org/10.1002/j.1538-7305.1959.tb01585.x.

[LY22] Li, Jiatu y Tianqi Yang: $3.1n - o(n)$ *Circuit Lower Bounds for Explicit Functions*. En *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, página 1180–1193. Association for Computing Machinery, 2022. https://doi.org/10.1145/3519935.3519976.

[RC20] Román Calvo, Enrique: *Estudio de la endogamia de los circuitos booleanos*, 2020. https://eprints.ucm.es/id/eprint/61715/, Trabajo de Fin de Grado en Ingeniería Informática y Matemáticas, Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid.

[RM22a] Rubio Madrigal, Celia: *Análisis de una red neuronal para la identificación de funciones booleanas complejas*. Trabajo de Fin de Grado en Ingeniería Informática, Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, 2022.

[RM22b] Rubio Madrigal, Celia: *Repositorio del presente trabajo*, 2022. https://github.com/celrm/tfg.git.

[RR97] Razborov, Alexander A. y Steven Rudich: *Natural Proofs*. Journal of Computer and System Sciences, 55(1):24–35, 1997. https://doi.org/10.1006/jcss.1997.1494.

[Spe87] Spearman, C.: *The Proof and Measurement of Association between Two Things*. The American Journal of Psychology, 100(3/4):441–471, 1987. https://doi.org/10.2307/1422689.

[SW93] Sieling, Detlef y Ingo Wegener: *Reduction of OBDDs in Linear Time*. Information Processing Letters, 48(3):139–144, 1993. https://doi.org/10.1016/0020-0190(93)90256-9.

[Tur36] Turing, Alan M.: *On Computable Numbers, with an Application to the Entscheidungsproblem*. Proceedings of the London Mathematical Society, s2-42(1):230–265, 1936. https://doi.org/10.1112/plms/s2-42.1.230.

[VE21] Villarrubia Elvira, Jorge: *Identificación experimental de las funciones booleanas que requieren circuitos extensos y aplicación al estudio de P vs NP*, 2021. https://eprints.ucm.es/id/eprint/66951/, Trabajo

de Fin de Grado en Ingeniería Informática, Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid.

[Wig07] Wigderson, Avi: *P, NP and Mathematics - A computational complexity perspective*. Proceedings of the International Congress of Mathematics, 1:665–712, 2007. https://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/W06/w06.pdf.