Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Matemáticas e Informática

Trabajo de Fin de Grado

Desarrollo de una aplicación para ilustrar contenidos de la asignatura Matemática Discreta I: Algoritmo de Quine-McCluskey

Autor: Ricardo Alberto Ferreiro de Aguiar
Tutor: Luis Magdalena Layos

Madrid, Junio-2024

Este Trabajo de Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo de Fin de Grado
Grado en Matemáticas e Informática

Título: Desarrollo de una aplicación para ilustrar contenidos de la asignatura Matemática Discreta I: Algoritmo de Quine-McCluskey

Junio-2024

Autor: Ricardo Alberto Ferreiro de Aguiar
Tutor: Luis Magdalena Layos
Matemática aplicadas a las tecnologías de la información y las comunicaciones
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Índice general

Índice de Tablas

Agradecimientos

A todos los profesores que he tenido por ofrecer sus conocimientos, especialmente mi tutor Luis Magdalena, por toda la ayuda brindada durante estos meses.

A todos mis amigos, por estar siempre a mi lado, y a mis nuevos amigos que he conocido en esta etapa, que siempre me han ayudado. Vuestro apoyo ha sido fundamental para mantenerme motivado y enfocado.

A mi madre y padre, por darme la oportunidad de estudiar lo que quería y ser mis referentes. A mis hermanos por apoyarme en todo momento. Vuestro amor y sacrificio han sido el pilar sobre el que he construido mis sueños.

Gracias de corazón, sin vuestro aliento y confianza, este logro no habría sido posible.

Resumen

Este Trabajo de Fin de Grado se engloba dentro del denominado ”Proyecto Calculadora” que se desarrolla como parte de la actividad del Grupo de Innovación Educativa ”Desarrollo de Tecnologías en la Enseñanza de las Matemáticas” en la Universidad Politécnica de Madrid.

Este proyecto se centra en el diseño e implementación de una página web que pueda servir como interfaz de cálculo para la resolución de problemas. Las librerías utilizadas para solucionarlos son TFGs que definen los métodos y clases necesarios. Estos problemas estarán relacionados con distintas asignaturas matemáticas cursadas en los grados que se imparten actualmente en la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Madrid.

Más detalladamente, este trabajo se enfoca en el algoritmo de Quine-McCluskey y su relación con los mapas de Karnaugh. El objetivo principal es proporcionar una solución práctica y efectiva para la enseñanza de la simplificación booleana, contribuyendo al avance de las herramientas educativas digitales y mejorando la calidad del aprendizaje en Matemática Discreta.

Palabras Clave: Algoritmo de Quine-McCluskey, Mapas de Karnaugh, Simplificación booleana, Educación digital, Matemática Discreta.

Abstract

This Final Degree Project is part of the ”Calculator Project” which is developed as part of the activity of the Educational Innovation Group ”Development of Technologies in the Teaching of Mathematics” at the Polytechnic University of Madrid.

This project focuses on the design and implementation of a web page that can serve as a calculus interface for problem solving. The libraries used to solve them are TFGs that define the necessary methods and classes. These problems will be related to different mathematical subjects studied in the degrees currently taught at the Escuela Técnica Superior de Ingeniería Informática of the Universidad Politécnica de Madrid.

In more detail, this work focuses on the Quine-McCluskey algorithm and its relation with Karnaugh maps. The main objective is to provide a practical and effective solution for teaching Boolean simplification, contributing to the advancement of digital educational tools and improving the quality of learning in Discrete Mathematics.

Keywords: Quine-McCluskey Algorithm, Karnaugh Maps, Boolean Simplification, Digital Education, Discrete Mathematics.

Capítulo 1

Introducción

1.1. Contexto del proyecto

Este Trabajo de Fin de Grado está enmarcado en el proyecto "Calculadora", una herramienta web didáctica diseñada para mejorar la comprensión y aplicación de conceptos matemáticos complejos a través de soluciones interactivas y visualmente intuitivas.

Para lograr su objetivo, el proyecto utiliza librerías que son TFGs que definen los métodos y clases necesarios para la resolución de los distintos problemas. Estos problemas están relacionados con las distintas asignaturas de matemáticas del Grado de Matemáticas e Informática que se imparten actualmente en la Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid. El departamento que dirige este proyecto es el Departamento de Matemáticas Aplicadas a las TIC (DMATIC). Este proyecto cuenta con componentes resolutivas que ya están en funcionamiento.

Este Trabajo de Fin de Grado se centra en la asignatura Matemática Discreta I, más específicamente en el algoritmo de Quine-McCluskey, además se verá su correspondecia con los mapas de Karnaugh. Se seleccionó debido a su relevancia y utilidad en la simplificación de funciones booleanas. A pesar de su importancia, muchos estudiantes encuentran dificultades en comprender y aplicar correctamente este algoritmo, lo que justifica la creación de herramientas didácticas específicas para su enseñanza.

"Calculadora" no sólo se alinea con la creciente demanda de herramientas educativas digitales que facilitan el aprendizaje autodidacta, sino que también responde a una necesidad académica de proporcionar apoyo adicional en áreas complejas.

1.2. Motivación del proyecto

La investigación sobre el algoritmo de Quine-McCluskey es esencial por varias razones que destacan su relevancia. En el ámbito de la Matemática Discreta, este algoritmo es una herramienta para la simplificación de funciones booleanas, un algoritmo importante en áreas como el diseño de circuitos digitales, la optimización de sistemas y la resolución de problemas lógicos. Debido a su impacto, entender y aplicar

este algoritmo es esencial para los estudiantes.

La motivación principal para realizar este trabajo nace de las dificultades que tienen los estudiantes para comprender el algoritmo de Quine-McCluskey. Pese a su importancia, los estudiantes encuentran el algoritmo complejo. Este proyecto pretende abordar estas dificultades desarrollando una herramienta interactiva que facilite el aprendizaje y el uso del algoritmo. La creación de esta herramienta no solo ayudará a los estudiantes a superar los obstáculos que se encuentran, sino que también enriquecerá el proceso educativo, haciéndolo más efectivo.

Además, en un contexto más amplio, el trabajo contribuye al avance científico y tecnológico, ya que proporiciona una implementación práctica del algoritmo en forma de una aplicación interactiva, se genera un recurso que puede ser utilizado tanto en entornos educativos como profesionales. Esta herramienta puede ser integrada en diversas plataformas de software, apoyando tanto la enseñanza como la investigación en matemáticas e informática.

1.3. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es implementar una herramienta, que sirve como ayuda y soporte para entender y afianzar los conocimientos recibidos en las clases de Matemática Discreta I sobre el algoritmo de Quine-McCluskey, haciendo un enfoque también en los mapas de Karnaugh. Otros de los objetivos son:

- Facilitar la comprensión y manejo del algoritmo a los alumnos que enfrentan dificultades en esta área. El diseño de la herramienta abordará las dificultades que tengan los estudiantes, proporcionando ejemplos útiles.

- Enriquecer la experiencia educativa a través de la integración de herramientas digitales que permitan una aplicación práctica y directa de los conceptos teóricos aprendidos en clase. Se ofrecerá un entorno de aprendizaje que permite ver a los estudiantes el procedimiento para la resolución de ejercicios, reforzando su comprensión de conceptos teóricos.

- Ampliación de habilidades y conocimientos en el desarrollo de aplicaciones web, con un enfoque particular en la creación de interfaces interactivas e intuitivas para los usuarios, manteniendo el diseño de la propia aplicación web del proyecto ”Calculadora”.

- Profundizar en la comprensión teórica y práctica de algoritmos, específicamente en el algoritmo de Quine-McCluskey, para mejorar la capacidad de resolver problemas y optimizar procesos computacionales. La idea de este Trabajo de Fin de Grado, además de implementar el algoritmo, es profundizar en su teoría subyacente.

- Integrar el código desarrollado en el marco de una aplicación web existente, ”Calculadora”, asegurando que cumple con las pautas y estándares establecidos. Esta integración permitirá una mayor funcionalidad y una experiencia de usuario más cohesiva.

1.4. Estructura del Documento

Este Trabajo de Fin de Grado está organizado en 6 capítulos, incluyendo la propia Introducción, que abordan distintos aspectos del algoritmo de Quine-McCluskey y su aplicación:

- **Marco Teórico**: Proporciona los fundamentos teóricos necesarios para entender el algoritmo de Quine-McCluskey y los conceptos relacionados. Incluye secciones sobre álgebra de Boole, expresiones booleanas, tablas de verdad, mapas de Karnaugh y ejemplos prácticos.
- **Código**: Describe el entorno de desarrollo y las librerías utilizadas, y proporciona una explicación detallada del código implementado para el algoritmo de Quine-McCluskey y los mapas de Karnaugh. Se incluyen funciones auxiliares y ejemplos de implementación.
- **Guía de uso**: Detalla los distintos escenarios en los que la herramienta desarrollada puede ser aplicada. Se incluyen tres casos de uso principales: la simplificación de funciones booleanas, la comparación de resultados obtenidos mediante el algoritmo de Quine-McCluskey con los obtenidos mediante mapas de Karnaugh, y el uso de ejemplos predeterminados para ilustrar el funcionamiento de la herramienta. Además, se hace una guía para que los estudiantes tengan un ejemplo de cómo usar la aplicación.
- **Impacto del Trabajo**: Examina el impacto del proyecto en el ámbito educativo y su contribución a los Objetivos de Desarrollo Sostenible (ODS). Se discute cómo la herramienta desarrollada puede mejorar la educación de calidad, el trabajo decente y el crecimiento económico, la innovación y la reducción de desigualdades.
- **Resultados y Conclusiones**: Analiza los resultados obtenidos del proyecto y su efectividad en la enseñanza del algoritmo de Quine-McCluskey. Se discuten las aplicaciones prácticas en el aula y las conclusiones personales del autor. También se proponen futuras líneas de trabajo y mejoras para la herramienta.

Cada capítulo está diseñado para ofrecer una visión integral del desarrollo del proyecto, desde los fundamentos teóricos hasta los resultados prácticos y su impacto en el ámbito educativo.

Capítulo 2

Marco Teórico

Para comprender el algoritmo de Quine-McCluskey, se revisarán las bases teóricas relacionadas con las álgebras de Boole, dualidad, propiedades básicas, expresiones booleanas y sus formas canónicas, hasta llegar a otras técnicas de minimización como los mapas de Karnaugh. Se utilizarán como referencias principales los textos de [1], [2], [3], [4] y [5].

2.1. Álgebras de Boole

El álgebra de conjuntos, el cálculo proposicional y los circuitos con interruptores tienen propiedades comunes y, sus operaciones, grandes analogías. Resulta, entonces, provechoso estudiar lo que tienen en común estos tres temas. Estas leyes comunes sirven para definir una estructura matemática abstracta denominada *Álgebra de Boole*, en honor del matemático George Boole (1815-1864).

Definición 1. Sea $\mathbb{B}$ un conjunto no vacío con dos operaciones binarias: $+$ y $\cdot$, una operación unaria $^-$; y dos elementos distintos: 0 y 1. Entonces $\mathbb{B}$ se denomina **álgebra de Boole** si los siguientes axiomas se cumplen, donde a, b, c son elementos arbitrarios en $\mathbb{B}$:

- [B_1] Leyes de conmutatividad:
  - (1a) $a + b = b + a$
  - (1b) $a \cdot b = b \cdot a$
- [B_2] Leyes de distributividad:
  - (2a) $a + (b \cdot c) = (a + b) \cdot (a + c)$
  - (2b) $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
- [B_3] Leyes de identidad:
  - (3a) $a + 0 = a$
  - (3b) $a \cdot 1 = a$
- [B_4] Leyes de complementariedad:
  - (4a) $a + \overline{a} = 1$
  - (4b) $a \cdot \overline{a} = 0$

Algunas veces, un álgebra de Boole se designa por $\langle \mathbf{B}, +, \cdot, ^-, 0, 1 \rangle$ cuando se quiere resaltar sus seis partes. Se dice que 0 es el elemento cero; 1, el elemento unidad y $\overline{a}$ el complemento de a. El símbolo $\cdot$ no suele usarse y en su lugar se usa la yuxtaposición.

Las operaciones $+$, $\cdot$ y $^-$ se denominan, respectivamente, suma, producto y complemento. Se adopta la convención de costumbre de que, a menos que haya paréntesis, $^-$ tiene precedencia sobre $\cdot$ y $\cdot$ tiene precedencia sobre $+$.

Ejemplo 1. Sea $\mathbf{B} = \{0, 1\}$ el conjunto de bits (dígitos binarios), con las operaciones binarias de $+$ y $\cdot$ definida por el cuadro 2.1. Supongamos que $\overline{1} = 0$ y $\overline{0} = 1$. Entonces $\mathbf{B}$ es un álgebra de Boole.

Cuadro 2.1: Tabla de operaciones de $\mathbf{B}$.

$+$	0	1
0	0	1
1	1	1

$\cdot$	0	1
0	0	0
1	0	1

Ejemplo 2. El álgebra de Boole $\langle \mathcal{P}(A), \cup, \cap, ^{-}, \emptyset, A\rangle$ de todos los subconjuntos de un conjunto no vacío A bajo las operaciones usuales de unión, intersección y complementación; y con $\emptyset$ y A como los elementos distinguidos 0 y 1.

Ejemplo 3. Sea $\mathbf{D}_{70} = \{1, 2, 5, 7, 10, 14, 35, 70\}$, los divisores de 70. Las operaciones $+, \cdot, ^{-}$ se definen sobre $\mathbf{D}_{70}$ como

$$a + b = mcm(a, b), \qquad a \cdot b = mcd(a, b), \qquad \overline{a} = \frac{70}{a}.$$

Entonces $\mathbf{D}_{70}$ es un álgebra de Boole con 1 como elemento cero y 70 como elemento unidad.

2.2. Dualidad

El dual de cualquier proposición en un álgebra de Boole $\mathbb{B}$ es la proposición que se obtiene al intercambiar las operaciones $+$ y $\cdot$, e intercambiar sus elementos identidad 0 y 1 en la proposición original. Por ejemplo, el dual de $(1+a)\cdot(b+0) = b$ es $(0\cdot a)+(b\cdot 1) = b$. Observamos la simetría en los axiomas de un álgebra de Boole $\mathbb{B}$. Es decir, el dual del conjunto de axiomas de $\mathbb{B}$ es el mismo que el conjunto original de axiomas. En consecuencia, se tiene el siguiente teorema.

Teorema 1. *(**Principio de dualidad**) El dual de cualquier teorema en un álgebra de Boole también es un teorema.*

En otras palabras, si cualquier proposición es una consecuencia de los axiomas de un álgebra de Boole, entonces el dual también es una consecuencia de estos axiomas, puesto que la proposición dual se demuestra al aplicar el dual de cada paso en la demostración de la proposición original.

2.3. Propiedades básicas

Las siguientes proposiciones se demuestras fácilmente mediante los axiomas [B_1] a [B_4].

Proposición 1. *Sean a, b, c elementos arbitrarios en un álgebra de Boole $\mathbb{B}$.*

1)	*Leyes de idempotencia:*		
	(5a)	$a + a = a$	*(5b)* $a \cdot a = a$
2)	*Leyes de acotamiento:*		
	(6a)	$a + 1 = 1$	*(6b)* $a \cdot 0 = 0$
3)	*Leyes de absorción:*		
	(7a)	$a + (a \cdot b) = a$	*(7b)* $a \cdot (a + b) = a$
4)	*Leyes asociativas:*		
	(8a)	$(a + b) + c = a + (b + c)$	*(8b)* $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
5)	*Leyes de De Morgan:*		
	(9a)	$\overline{a + b} = \overline{a} \cdot \overline{b}$	*(9b)* $\overline{a \cdot b} = \overline{a} + \overline{b}$
6)	*(10a)*	$\overline{0} = 1$	*(10b)* $\overline{1} = 0$

Proposición 2. *Sea a un elemento de un álgebra de Boole $\mathbb{B}$.*

1. *(Unicidad del complemento) Si $a + x = 1$ y $a \cdot x = 0$, entonces $\overline{a}$.*
2. *(Ley de involución) $\overline{\overline{a}} = a$.*

2.4. Expresiones booleanas y funciones booleanas

Consideremos un conjunto de variables (o letras o símbolos), por ejemplo, $x_1, x_2, \ldots, x_n$. Una **expresión booleana** E en estas variables, que escribiremos $E(x_1, x_2, \ldots, x_n)$, es cualquier variable o cualquier expresión que se obtiene a partir de las variables mediante las operaciones booleanas $+, \cdot, ^-$. Por ejemplo,

$$E = ((x\overline{y}\overline{z} + \overline{y}) + \overline{x}z)$$

es una expresión booleana en x, y, z.

Llamamos $\mathbb{B}^n$ al producto cartesiano de n copias de $\mathbb{B}$, siendo $0 \neq n \in \mathbb{N}$.

Definición 2. Una **función booleana** de grado n es una aplicación de $\mathbb{B}^n$ en $\mathbb{B}$. Los valores de una función booleana $f : \mathbb{B}^n \to \mathbb{B}$ son 0 o 1, y se representan en la forma $f(x_1, x_2, \ldots, x_n)$. Llamamos a cada x_i una variable booleana de f. Observar que cada x_i puede tomar sólo los valores 0 y 1.

Ejemplo 4. La función booleana $f : \mathbb{B}^n \to \mathbb{B}$ definida por $f(x, y, z) = xy + x\overline{z}$.

Dos funciones booleanas $f, g : \mathbb{B}^n \to \mathbb{B}$ son iguales si toman los mismos valores, esto es, si para cualquier n–upla $(b_1, b_2, \ldots, b_n) \in \mathbb{B}^n$ se tiene $f(b_1, b_2, \ldots, b_n) = f(b_1, b_2, \ldots, b_n)$.

2.5. Formas canónicas de funciones booleanas

Se trata de ver que toda función booleana se puede escribir como una expresión booleana y que, por lo tanto, el problema de estudiar funciones booleanas se reduce a estudiar, salvo equivalencia, expresiones booleanas.

Definición 3. Un **literal** es una variable booleana o el complemento de una variable booleana, como $x, \overline{x}, y, \overline{y}$. Un **producto fundamental** es un literal o un producto de dos o más literales donde ningún par de literales implican la misma variable. Así, $x\overline{z}$,

$x\overline{y}z$, x, $\overline{y}$, $\overline{x}yz$ son productos fundamentales, pero $xy\overline{x}z$ y $xyzy$ no lo son. Observemos que cualquier producto de literales puede reducirse a 0 o a un producto fundamental; por ejemplo, $xy\overline{x}z = 0$, puesto que $x\overline{x} = 0$ (ley de complemento), y $xyzy = xyz$ porque $yy = y$ (ley de idempotencia).

Un producto fundamental P_1 está contenido en (o incluido en) otro producto fundamental P_2 si las literales de P_1 también son las literales de P_2. Por ejemplo, $\overline{x}z$ está contenido en $\overline{x}yz$, pero $\overline{x}z$ no está contenido en $x\overline{y}z$ porque $\overline{x}$ no es una literal de $x\overline{y}z$. Además, si P_1 está contenido en P_2, por ejemplo, $P_2 = P_1 \cdot Q$, entonces, por la ley de absorción,

$$P_1 + P_2 = P_1 + P_1 \cdot Q = P_1.$$

Así, por ejemplo, $\overline{x}z + \overline{x}yz = \overline{x}z$.

Definición 4. Una expresión booleana E se denomina **expresión de suma de productos** si E es un producto fundamental o la suma de dos o más productos fundamentales, ninguno de los cuales está contenido en el otro.

Definición 5. Un **minitérmino** en la variables booleanas $x_1, x_2, \ldots, x_n$ es un producto booleano $y_1 y_2 \cdots y_n$, donde $y_i = x_i$ o $y_i = \overline{x}_i$. Es decir, un minitérmino es un producto de n literales con un literal por cada variable.

Se observa que un minitérmino vale 1 para una y sólo una combinación de sus variables. En concreto, el minitérmino $y_1 y_2 \cdots y_n$ es 1 si y sólo si cada $y_i = 1$, y esto ocurre si y sólo si $x_i = 1$ cuando $y_i = x_i$, y $x_i = 0$ cuando $y_i = \overline{x}_i$.

Ejemplo 5. Si tenemos las variables x_1, x_2, x_3, x_4, son minitérminos: $x_1\overline{x}_2\overline{x}_3x_4$, $x_1x_2x_3\overline{x}_4$ y $x_1x_2x_3x_4$. En cambio no lo son: x_1, x_2, x_3, x_4, $x_1x_2\overline{x}_4$, $x_1x_2x_3x_4x_5$.

Ejemplo 6. El minitérmino que toma el valor 1 para los valores de las variables siguientes: $x_1 = x_3 = x_5 = 1$ y $x_2 = x_4 = x_6 = 0$, corresponde a $x_1\overline{x}_2x_3\overline{x}_4x_5\overline{x}_6$.

Definición 6. Sea E una expresión booleana arbitraria. Una **forma de suma de productos** de E es una expresión de suma de productos booleana equivalente.

Ejemplo 7. Consideremos las expresiones

$$E_1 = x\overline{z} + \overline{y}z + xy\overline{z} \quad \text{y} \quad E_2 = x\overline{z} + \overline{x}yz + xz.$$

Aunque la primera expresión, E_1, es una suma de productos, no es una expresión de suma de productos. En concreto, el producto $x\overline{z}$ está contenido en el producto $xy\overline{z}$. Sin embargo, por la ley de absorción, E_1 se expresa como

$$E_1 = x\overline{z} + \overline{y}z + xy\overline{z} = x\overline{z} + xy\overline{z} + \overline{y}z = x\overline{z} + \overline{y}z.$$

Esto lleva a una forma de suma de productos para E_1. La segunda expresión E_2 ya es una expresión de suma de productos.

Algoritmo para encontrar formas de suma de productos. Se proporciona un algoritmo de cuatro pasos que aplica leyes del álgebra de Boole para transformar cualquier expresión booleana en una expresión de suma de productos equivalente.

Algoritmo:	La entrada es una expresión booleana E. El resultado es una expresión de suma de productos equivalente a E.
Paso 1.	Se usan las leyes de De Morgan y de involución para mover la operación complemento en cualquier paréntesis hasta que la operación complemento sólo se aplica a variables. Entonces, E sólo consistirá de sumas y productos de literales.
Paso 2.	Se usa la operación distributiva para transformar a continuación E en una suma de productos.
Paso 3.	Se usan las leyes conmutativa, de idempotencia y de complemento para transformar cada producto en E en 0 o en un producto fundamental.
Paso 4.	Se usan las leyes de absorción e identidad para transformar finalmente E en una expresión de suma de productos.

Cuadro 2.2: Algoritmo para encontrar formas de suma de productos.

Ejemplo 8. Supongamos que el algoritmo del cuadro 2.5 se aplica a la siguiente expresión booleana:

$$E = (\overline{\overline{x}\overline{y}z})(\overline{\overline{x}+z)(\overline{y}+z)}$$

Paso 1. Al usar las leyes de De Morgan y de involución se obtiene

$$E = (x\overline{\overline{y}} + \overline{z})((\overline{\overline{x}+z}) + (\overline{\overline{y}+\overline{z}})) = (xy + \overline{z})(x\overline{z} + yz).$$

Ahora E consiste sólo de sumas y productos de literales.

Paso 2. Al usar las leyes distributivas, se obtiene

$$E = xyx\overline{z} + xyyz + x\overline{z}\overline{z} + yz\overline{z}.$$

Ahora E es una suma de productos.

Paso 3. Al usar las leyes conmutativa, de idempotencia y de complemento, se obtiene

$$E = xy\overline{z} + xyz + x\overline{z} + 0.$$

Cada término en E es un producto fundamental o 0.

Paso 4. El producto $x\overline{z}$ está contenido en $xy\overline{z}$; así, por la ley de absorción,

$$x\overline{z} + (x\overline{z}y) = x\overline{z}$$

Entonces, es posible borrar $xy\overline{z}$ de la suma. También, por la ley identidad para 0, es posible eliminar 0 de la suma. En consecuencia,

$$E = xyz + x\overline{z}$$

Ahora, E está representada por una expresión de suma de productos.

Definición 7. Se dice que una expresión booleana $E = E(x_1, x_2, \ldots, x_n)$ es una **forma completa de suma de productos** o bien una **forma normal disyuntiva** si E es una suma de minitérminos.

De esta forma, dada una función booleana, se puede construir una suma booleana de minitérminos que valga 1 cuando esta función booleana vale 1 y que valga 0 cuando la función tome el valor 0. El siguiente teorema es válido.

Teorema 2. *Cualquier expresión booleana $E = E(x_1, x_2, \ldots, x_n)$ diferente de cero es equivalente a una forma normal disyuntiva y esta representación es única.*

Demostración. Sea E una expresión booleana arbitraria diferente de 0 de las n variables $(x_1, x_2, \ldots, x_n)$. Si E contiene una expresión de la forma $\overline{r+r}$ o $\overline{rs}$ para ciertas variables r y s, se pueden aplicar las leyes de De Morgan para obtener $\overline{rs}$ y $\overline{r}+\overline{s}$, respectivamente. Este proceso puede continuarse hasta que cada complemento que aparece se aplica sólo a una única variable x_i. A continuación, aplicando la ley distributiva de $\cdot$ sobre $+$, E puede reducirse a un polinomio. Supongamos ahora que algún término no contiene ni x_i ni $\overline{x_i}$. Este término puede multiplicarse por $(x_i + \overline{x_i})$ produciendo una expresión equivalente. Continuando este proceso para cada variable que falte en cada uno de los términos de E, se obtendrá una expresión equivalente cuyos términos contengan x_i o $\overline{x_i}$ para cada $j = l, 2, \ldots, n$. Finalmente, las leyes de idempotencia permiten eliminar los términos duplicados.

■

El algoritmo en el cuadro 2.5 indica cómo transformar E en una forma de suma de productos. El cuadro 2.3 siguiente contiene un algoritmo que transforma una forma de suma de productos en una forma normal disyuntiva (forma completa de suma de productos).

Algoritmo:	La entrada es una expresión booleana E de suma de productos $E = E(x_1, x_2, \ldots, x_n)$. El resultado es una expresión completa de suma de productos equivalente a E.
Paso 1.	Se encuentra un producto P en E que no implique a la variable x_i y luego P se multiplica por $x_i + \overline{x}_i$, eliminando todos los productos repetidos. (Esto es posible puesto que $x_i + \overline{x}_i = 1$ y $P + P = P$)
Paso 2.	Se repite el paso 1 hasta que todo producto en E es un minitérmino; es decir, que todo producto P implica a todas las variables.

Cuadro 2.3: Algoritmo para encontrar formas normales disyuntivas.

Ejemplo 9. Sea $E(x, y, z) = x(\overline{\overline{y}z})$.

a) El algoritmo en el cuadro 2.5 se aplica a E, de modo que E quede representada como una expresión de suma de productos:

$$E = x(\overline{\overline{y}z}) = x(y + \overline{z}) = xy + x\overline{z}.$$

b) Luego se aplica el algoritmo en el cuadro 2.3 para obtener:

$$\begin{aligned} E &= xy(z + \overline{z}) + x\overline{z}(y + \overline{y}) \\ &= xyz + xy\overline{z} + xy\overline{z} + x\overline{yz} \\ &= xyz + xy\overline{z} + x\overline{yz}. \end{aligned}$$

Ahora, E está representada en su forma normal disyuntiva.

2.6. Expresiones booleanas minimales. Implicantes primos.

Hay muchas formas de representar la misma expresión booleana E. En esta sección se define e investiga una forma de suma de productos *minimal* para E. Será necesario definir los *implicantes primos* de E puesto que la suma de productos minimal supone tales implicantes. En la literatura se pueden encontrar otras formas minimales.

El problema de la simplificación de una función booleana, tal como lo formuló Quine [6], puede plantearse de la siguiente manera: dada una función booleana (completamente definida) f, encontrar las formas disyuntivas (y/o conjuntivas) más sencillas que sean equivalentes a f. Dado que el diseño económico de circuitos es un factor importante en la síntesis de un sistema de conmutación, este problema ha recibido mucha atención en los últimos años y en la actualidad se pueden encontrar varias soluciones disponibles (véase [7]).

Suma de productos minimal. Consideremos una expresión booleana de suma de productos E. Sea E_L el número de literales en E (contadas según multiplicidad), y sea E_S el número de sumandos en E. Por ejemplo, supongamos

$$E = xy\overline{z} + \overline{x}\overline{y}t + x\overline{y}\overline{z}t + \overline{x}yzt.$$

Entonces, $E_L = 3 + 3 + 4 + 4 = 14$ y $E_S = 4$.

Supongamos que E y F son expresiones booleanas de suma de productos equivalentes. Se dice que E es más **simple** que F si:

$$i)\ \ E_L < F_L \ \text{ y } \ E_L < F_S, \ \text{ o } \ \ ii)\ \ E_L \leq F_L \ \text{ y } \ E_L < F_S.$$

Se dice que E es **minimal** si no hay ninguna expresión de suma de productos más simple que E. Obsérvese que puede haber más de una expresión minimal de suma de productos equivalente.

Definición 8. Dadas dos funciones booleanas f y g en $\mathbb{B}^n$, decimos que f **implica** a g (o que f es un *minorante* de g, o que g es un *mayorante* de f) si

$$f(x_1, x_2, \ldots, x_n) = 1 \implies g(x_1, x_2, \ldots, x_n) = 1 \ \text{ para todo } \ (x_1, x_2, \ldots, x_n) \in \mathbb{B}^n.$$

Cuando este es el caso, escribimos $f \leq g$.

Esta definición se extiende de forma directa a las expresiones booleanas, ya que toda expresión de este tipo puede considerarse una función booleana. La terminología «f implica g» está obviamente tomada de la lógica: si f y g representan, respectivamente, el valor de verdad de los enunciados proposicionales S_f y S_g, entonces $f \leq g$ se cumple exactamente cuando $S_f \Rightarrow S_g$.

Las siguientes formulaciones alternativas de la definición 8 suelen ser útiles.

Teorema 3. *Para todas las funciones booleanas f y g sobre $\mathbb{B}^n$, las siguientes afirmaciones son equivalentes:*

(1) $f \leq g$;

(2) $f \vee g = g$;

(3) $\overline{f} \vee g = \mathbf{1}_n$;

(4) $fg = f$;

(5) $f\overline{g} = \mathbf{0}_n$.

Demostración. Basta observar que cada una de las afirmaciones (1)–(5) falla exactamente cuando existe $(x_1, x_2, \ldots, x_n) \in \mathbb{B}^n$ tal que $f(x_1, x_2, \ldots, x_n) = 1$ y $g(x_1, x_2, \ldots, x_n) = 0$.

■

Cuando dos funciones booleanas f y g están representadas por expresiones booleanas arbitrarias, puede ser bastante difícil comprobar si f implica o no a g. La definición 8 no sugiere ninguna forma eficiente de realizar esta tarea, excepto mediante la enumeración completa de todos los puntos de $\mathbb{B}^n$, ni el teorema 3 ayuda en este sentido. Para los productos fundamentales, sin embargo, la implicación adopta una forma especialmente sencilla y fácilmente verificable: en efecto, un producto fundamental implica a otro si y sólo si este último resulta de la primera por supresión de literales (el producto fundamental «más largo» implica al «más corto»). Más formalmente:

Teorema 4. *El producto fundamental* $P_{AB} = \prod_{i\in A} x_i \prod_{j\in B} \overline{x}_j$ *implica el producto fundamental* $P_{FG} = \prod_{i\in F} x_i \prod_{j\in G} \overline{x}_j$ *si y sólo si* $F \subseteq A$ *y* $G \subseteq A$.

Demostración. Denotamos por $X = (x_1, x_2, \ldots, x_n)$ un punto arbitrario en $\mathbb{B}^n$. Supongamos que $F \subseteq A$ y $G \subseteq A$ y consideremos cualquier punto $X \in \mathbb{B}^n$. Si $P_{AB} = 1$, entonces $x_i = 1$ para todo $i \in A$ y $x_j = 0$ para todo $j \in B$, de modo que $x_i = 1$ para todo $i \in F$ y $x_j = 0$ para todo $j \in G$. Por tanto, $P_{FG} = 1$ y concluimos que P_{AB} implica P_{FG}.

Recíprocamente, supongamos por ejemplo que F no está contenido en A. Fijemos $x_i = 1$ para todo $i \in A$, $x_j = 0$ para todo $j \notin A$ y X. Entonces, $P_{AB} = 1$ pero $P_{FG} = 0$ (ya que $x_k = 0$ para algún $k \in F \setminus A$), por lo que P_{AB} no implica P_{FG}.

■

Definición 9. Sea f una función booleana y P un producto fundamental. Decimos que P es un **implicante** de f si P implica f.

Ejemplo 10. Sea $f = xy + x\overline{y}z$. Entonces xy, $x\overline{y}z$ y xz son implicantes de f.

Teorema 5. *Si* E *es una forma normal disyuntiva de la función booleana* f*, entonces cada término de* E *es un implicante de* f*. Además, si* P *es un implicante de* f*, entonces la forma normal disyuntiva* $P + E$ *también representa a* f*.*

Demostración. Para la primera afirmación, obsérvese que, si algún término de E toma valor 1, entonces E, y por tanto f, toman valor 1. Para la segunda afirmación, basta con comprobar sucesivamente que $P + E \leq f$ y $f \leq E \leq P + E$.

■

Ejemplo 11. Por el teorema 5, la función $f = xy + x\overline{y}z$ (véase el ejemplo 10) admite la forma normal disyuntiva $xy + x\overline{y}z + xz = xy + xz$ (la última igualdad se verifica fácilmente).

El ejemplo 11 ilustra un punto importante: con vistas a la simplificación de las expresiones booleanas, tiene sentido sustituir los implicantes «largos» por los «cortos» en el DNF de una función booleana. El significado de «largos» y «cortos» puede aclararse consultando el teorema 4. Esta línea de razonamiento conduce a las siguientes definiciones (véase Quine [6]).

Definición 10. Un producto fundamental P se denomina **implicante primo** de una expresión booleana E si

$$P + E = E \text{ o, equivalentemente, } P \leq E$$

pero ningún otro producto fundamental contenido en P tiene esta propiedad. En otras palabras, si E es un implicante de f y $P \leq E$, entonces $P = E$.

Ejemplo 12. Consideremos de nuevo la función f definida en el ejemplo 10. Es fácil comprobar que xy y xz son implicantes primos de f , mientras que $x\overline{y}z$ no es primo (ya que $x\overline{y}z \leq xz$). De hecho, f no tiene otros implicantes primos que xy y xz.

Los implicantes primos desempeñan un papel crucial en la construcción de las formas normales disyuntivas de funciones booleanas. Este papel se describe mejor en el siguiente teorema:

Teorema 6. *Toda función booleana puede representarse por la suma de todos sus implicantes primos.*

Demostración. Sea f una función booleana sobre $\mathbb{B}^n$, y sean $P_1, P_2, \ldots P_m$ sus implicantes primos (nótese que m es finito porque el número de productos elementales sobre n variables es finito). Consideremos cualquier forma normal disyuntiva de f, digamos $E = \sum_{k=1}^{r} C_k$. Por el teorema 5, la forma normal disyuntiva

$$E' = \sum_{k=1}^{r} C_k + \sum_{j=1}^{m} P_j$$

también representa a f. Consideremos cualquier término C_k de E ($1 \leq k \leq r$). Puesto que C_k es un implicante de f, existe al menos un implicante primo P_j con $C_k \leq P_j$ (donde posiblemente $C_k = P_j$). Entonces, $C_k + P_j = P_j$, de lo que se deduce $E' = \sum_{j=1}^{m} P_j$.

■

Obsérvese que la suma de todos los implicantes primos de una función booleana sólo es única salvo el orden de sus términos y literales. Sin embargo, al igual que hicimos en el caso de las expresiones de minitérminos, haremos caso omiso de esta sutileza y simplemente consideraremos tal suma como unívoca.

Un corolario interesante del teorema 6 es que cada función booleana está identificada unívocamente por la lista de sus implicantes primos. Equivalentemente, dos

funciones booleanas son iguales si y sólo si tienen la misma suma de todos los implicantes primos. Subrayemos, sin embargo, que no siempre es necesario conocer todos los implicantes primos de una función para conocer la función, y que no siempre es necesario tomar la suma de todos los implicantes primos para obtener una representación normal disyuntiva correcta de la función.

Ejemplo 13. Sea $E = x\overline{y} + xy\overline{z} + \overline{x}y\overline{z}$. Puesto que la suma de productos es única, $A + E = E$, donde $A \neq 0$ si y sólo si los sumandos de la forma normal disyuntiva de A están entre los sumandos de la forma normal disyuntiva de E. Por tanto, primero hallamos la forma normal disyuntiva de E:

$$E = x\overline{y}(z + \overline{z}) + xy\overline{z} + \overline{x}y\overline{z} = x\overline{y}z + x\overline{yz} + xy\overline{z} + \overline{x}y\overline{z}.$$

a) Expresamos $x\overline{z}$ en la forma normal disyuntiva:

$$x\overline{z} = x\overline{z}(y + \overline{y}) = xy\overline{z} + x\overline{yz}.$$

Puesto que los sumandos de $x\overline{z}$ están entre los sumandos de E, tenemos que $x\overline{z} + E = E$.

b) Expresamos x en la forma normal disyuntiva:

$$x = x(y + \overline{y})(z + \overline{z}) = xyz + xy\overline{z} + x\overline{y}z + x\overline{yz}.$$

El sumando xyz de x no es un sumando de E; por tanto, $x + E \neq E$.

c) Expresamos $\overline{z}$ en la forma normal disyuntiva:

$$\overline{z} = \overline{z}(x + \overline{x})(y + \overline{y}) = xy\overline{z} + x\overline{yz} + \overline{x}y\overline{z} + \overline{xyz}.$$

El sumando $\overline{xyz}$ de $\overline{z}$ no es un sumando de E; por tanto, $\overline{z} + E \neq E$.
De esta forma, $x\overline{z}$ es un implicante primo de E.

Ejemplo 14. La función $g = x\overline{y} + \overline{x}y + x\overline{z}$ tiene cuatro implicantes primos, a saber, $x\overline{y}$, $\overline{x}y$, $x\overline{z}$ e $y\overline{z}$.

El siguiente teorema es válido.

Teorema 7. *Toda forma normal disyuntiva minimal M de una expresión booleana E es una suma de implicantes primos de E.*

Demostración. Sea C un producto fundamental de M. Si C no fuese un implicante primo[1] de E, entonces C incluiría un producto fundamental C' tal que C' implica a E. Si construimos M^* a partir de M reemplazando C por C', entonces M^* es equivalente a M: en efecto, por un lado, C implica C' y, en consecuencia, M implica M^*. Por otro lado, C' implica a E, el cual es equivalente a M y, por lo tanto, M^* implica a M. Pero M^* es más simple que M, contradiciendo la asunción que M es una forma normal minimal.

[1]Todo producto fundamental que implica a E debe necesariamente incluir un implicante primo de E. En efecto, si el producto fundamental C no es un implicante primo de E, entonces C debe incluir un subproducto fundamental C_1 el cual implica a E. Si C_1 no fuese un implicante primo de E, entonces C_1 debe incluir un subproducto fundamental C_2 el cual implica a E, y así sucesivamente. Este procedimiento debe detenerse finalmente, dando lugar a un implicante primo de E que está incluido en C.

■

Observación 1. Una forma normal disyuntiva minimal para E no tiene por qué ser la suma de *todos* los implicantes primos de E.

De manera más general, introduzcamos la siguiente terminología.

Definición 11. Sea f una función booleana sobre $\mathbb{B}^n$ y sea $C = \sum_{k\in\Omega} C_k$ una forma normal disyuntiva de f. Decimos que C es una forma normal disyuntiva **irredundante** de f si no hay ningún $j \in \Omega$ tal que $\sum_{k\in\Omega\setminus\{j\}} C_k$ represente a f; en caso contrario, decimos que f es **redundante**.

Así, una forma normal disyuntiva redundante puede convertirse en una forma normal disyuntiva equivalente más corta eliminando algunos de sus términos. El ejemplo 14 muestra que la forma normal disyuntiva completa con todos sus sumandos implicantes primos de una función booleana no es necesariamente irredundante. Del mismo modo, si en una forma normal disyuntiva no todo sumando es un implicante primo, al menos uno de sus términos puede sustituirse por un implicante primo que lo *absorba* (recuérdese el teorema 5 y los comentarios que le siguen). Por lo tanto, las formas normales disyuntivas irredundantes, con todos sus sumandos implicantes primos, proporcionan las representaciones de formas normales disyuntivas más cortas posibles de las funciones booleanas.

Definición 12. Se llama **implicante primo esencial** a un implicante primo que contiene uno o más minitérminos que no están incluidos en cualquier otro implicante primo.

2.7. Tablas de verdad

Toda expresión booleana E define una función de verdad. Para los valores que puede tomar cada variable, es decir, puede tomar los valores 0 ó 1, podemos calcular su correspondiente valor en E. Este cálculo puede ser mostrado en una tabla de verdad.

Se puede observar que la tabla de verdad de una expresión booleana $E = E(x_1, x_2, \ldots, x_n)$ con n variables también puede verse como una función booleana de $\mathbb{B}^n$ a $\mathbb{B}$. Es decir, la tabla de la verdad de una función booleana f es una tabla en la que se representan todos los elementos de $\{0,1\}^n$ y sus imágenes:

x_1	x_2	...	x_{n-1}	x_n	$f(x_1, x_2, ..., x_n)$
0	0	...	0	0	$f(0,0,...,0,0,0)$
0	0	...	0	1	$f(0,0,...,0,0,1)$
0	0	...	1	0	$f(0,0,...,0,1,0)$
0	0	...	1	1	$f(0,0,...,0,1,1)$
⋮	⋮	...	⋮	⋮	⋮
1	1	...	1	1	$f(1,1,...,1,1,1)$

Cada fila corresponde con los valores que toman las expresiones booleanas. Las columnas dan los valores de verdad correspondientes a las expresiones de suma de

productos que aparecen en la construcción paso a paso de la expresión booleana dada.

Ejemplo 15. Sea $f : B^2 \to B$, donde $f(x, y) = x\bar{y}$. Esta expresión booleana se muestra en la tabla de la verdad siguiente:

x	y	$\bar{y}$	$x\bar{y}$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

Ejemplo 16. Sea $f : B^3 \to B$, donde $f(x, y, z) = \bar{x} + yz$. Esta expresión booleana se muestra en la tabla de la verdad siguiente:

x	y	z	$\bar{x}$	yz	$\bar{x} + yz$
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	1

En este ejemplo, los minitérminos serían las filas: 1, 2, 3, 8, que serían los minitérminos en decimal. Es decir, $f = \sum m(1, 2, 3, 8)$.

Cuando tenemos tres literales en la expresión booleana, se observa que la tabla de verdad tiene ocho filas. En general, cuando hay n literales, hay 2^n filas en la tabla de verdad, dado que hay dos posibilidades, 0 ó 1, para cada literal.

2.8. Mapas de Karnaugh

Los mapas de Karnaugh es un mecanismo gráfico que se utiliza para encontrar implicantes primos y formas minimales para expresiones booleanas con un máximo de seis variables. En este trabajo se han considerado funciones lógicas de dos, tres y cuatro variables para trabajar con mapas de Karnaugh. El número de celdas del mapa de Karnaugh será 2^n, siendo n el número de variables.

Para la resolución a través de mapas de Karnaugh es necesario definir el concepto de productos adyacentes:

Proposición 3. *Dados dos minitérminos, se dice que son adyacentes si se cumple que:*

1. *Tienen igual número de literales.*
2. *Los literales corresponden en ambos términos a las mismas variables.*

3. *En ambas expresiones, todas menos un literal coinciden entre sí.*

En particular, la suma de estos dos productos adyacentes es un minitérmino con un literal menos.

Ejemplo 17. La suma de estos dos productos adyacentes P_1 y P_2, donde:

- $P_1 = xy\bar{z}$ y $P_2 = x\bar{y}\bar{z}$

$$P_1 + P_2 = xy\bar{z} + x\bar{y}\bar{z} = x\bar{z}(y` + bary) = x\bar{z}(1) = x\bar{z}$$

- $P_1 = xy\bar{z}$ y $P_2 = x\bar{y}z$ En este caso, P_1 y P_2 no son adyacentes, puesto que difieren en dos literales.
- $P_1 = xy\bar{z}$ y $P_2 = x\bar{y}\bar{z}t$ En este caso, P_1 y P_2 no son adyacentes, puesto que tienen variables diferentes.

Para la simplificación de funciones lógicas por medio de mapas de Karnaugh se realizan los siguientes pasos:

Algoritmo:	Se dibuja una tabla de Karnaugh con tantas casillas como 2^n , siendo n el número de variables.
Paso 1.	Marcamos las celdas que sean minitérminos, es decir, se marcará con un 1 si el término entrega como salida 1, o 0 si el minitérmino entrega como salida 0.
Paso 2.	Se deben agrupar 1s en grupos de 1,2,4,8,..., es decir, 2^n, y además la agrupación debe ser la mayor posible, sin importar que 1 o varios 1s pertenezcan a las mismas agrupaciones. Los grupos sólo se pueden hacer de forma horizontal y/o vertical.
Paso 3.	Después de obtener la agrupación de todos los elementos marcados, se procede a construir la función simplificada. Cada grupo de 1s corresponde a un término en la nueva función simplificada, que se obtiene multiplicando las variables presentes en ese grupo. Estos términos se suman para obtener la función simplificada completa.

Cuadro 2.4: Algoritmo para encontrar minitérminos.

Para obtener cada término de la función, se selecciona un grupo de 1s y se verifica si alguna variable dentro de ese grupo cambia de valor, es decir, pasa de 0 a 1 (o viceversa). En caso afirmativo, esas variables se eliminan del término correspondiente al grupo de 1s, sustituyendolas por un guión "-". Las variables que no cambian de valor y la variable "-" se multiplican entre sí y forman uno de los términos que se sumarán para obtener la función simplificada final. Este proceso se repite para cada grupo presente en la tabla, generando tantos términos como agrupaciones de 1s haya.

Finalmente, se suman todos los términos obtenidos para construir la función simplificada.

2.8.1. Caso de dos variables

El mapa de Karnaugh correspondiente a las expresiones booleanas $E = E(x, y)$ con dos variables x e y aparece en la Figura 2.1. Se puede observar que los cuadrados en la parte superior del mapa representan a x, y los cuadrados en la parte inferior a $\bar{x}$. Lo mismo pasa para la otra variable, en este caso los cuadrados en la parte izquierda del mapa representan a y, y los cuadrados de la parte derecha a $\bar{y}$. En consecuencia, los cuatro minitérminos posibles con dos literales

$$xy, x\bar{y}, \bar{x}y, \bar{x}\bar{y}$$

están representados por los cuatro cuadrados del mapa, como se observa en la figura. Observamos que dos de estos cuadrados son adyacentes si y solo si los cuadrados son geométricamente adyacentes (tienen un lado en común).

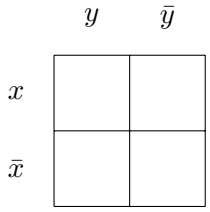
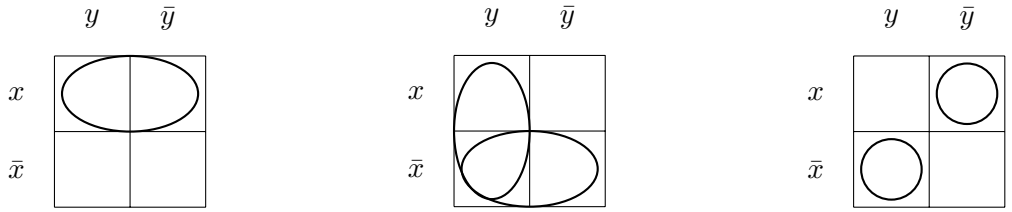


Figura 2.1: Mapa de Karnaugh.

Cualquier expresión booleana completa de suma de productos $E(x, y)$ es una suma de minitérminos y, por tanto, se representa en el mapa de Karnaugh mediante la señalización del cuadrado apropiado. Un implicante primo de $E(x, y)$ es un par de cuadrados adyacentes en el mapa o un cuadrado aislado, es decir, un cuadrado que no es adyacente a ningún otro. Una forma de suma de productos minimal para $E(x, y)$ consiste de un número mínimo de implicantes primos que cubran todos los cuadrados de $E(x, y)$.

Ejemplo 18. Sean $E_1 = xy + x\bar{y}$, $E_2 = xy + \bar{x}\bar{y} + \bar{x}\bar{y}$ y $E_3 = \bar{x}y + x\bar{y}$, como se observa en la Figura 2.2 encontramos los implicantes primos y una forma de suma de productos minimal para cada una de las expresiones booleanas.



(a) $E_1 = x$ (b) $E_2 = \bar{x} + y$ (c) $E_3 = \bar{x}y + x\bar{y}$

Figura 2.2: Mapas de Karnaugh de las expresiones booleanas dadas.

2.8.2. Caso de tres variables

El mapa de Karnaugh correspondiente a las expresiones booleanas $E = E(x, y, z)$ con tres variables x, y y z aparece en la Figura 2.3. En este caso habría ocho minitérminos

posibles con tres literales:

$$xyz, xy\bar{z}, x\bar{y}z, x\bar{y}\bar{z}, \bar{x}yz, \bar{x}y\bar{z}, \bar{x}\bar{y}z, \bar{x}\bar{y}\bar{z}$$

están representados por los ocho cuadrados del mapa, como se observa en la figura.

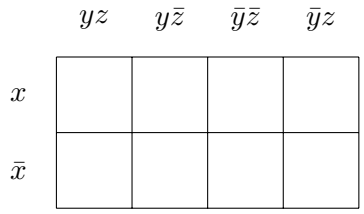
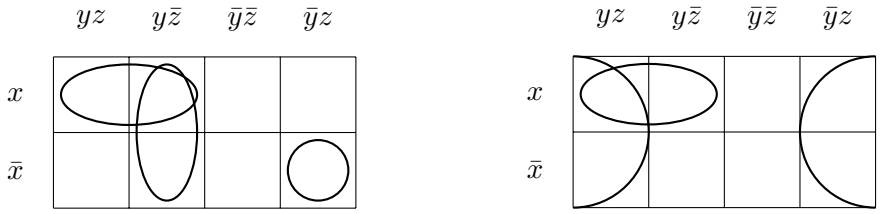


Figura 2.3: Mapa de Karnaugh de tres variables.

Entendemos un rectángulo básico en el mapa de Karnaugh con tres variables como un cuadrado, dos cuadrados adyacentes o cuatro cuadrados que forman un rectángulo de uno por cuatro o dos por dos. Estos rectángulos básicos corresponden a productos fundamentales de tres, dos y un literal, respectivamente. Además, el producto fundamental representado por un rectángulo básico es el producto de justo aquellos literales que aparecen en cada cuadrado del rectángulo.

Un implicante primo de E es un rectángulo básico maxima de E, es decir, un rectángulo básico contenido en E que no está contenido en ningún rectángulo básico más grande que E. Una forma de suma de productos minimal para E consiste de una cubierta minimal de E, es decir, un número minimal de rectángulos básicos maximales de E que juntos incluyen a todos los cuadrados de E.

Ejemplo 19. Sean $E_1 = xyz + xy\bar{z} + \bar{x}y\bar{z} + \bar{x}\bar{y}z$ y $E_2 = xyz + xy\bar{z} + x\bar{y}z + \bar{x}yz + \bar{x}\bar{y}z$, como se observa en la Figura 2.4 encontramos los implicantes primos y una forma de suma de productos minimal para cada una de las expresiones booleanas.



(a) $E_1 = xy + y\bar{z} + \bar{x}\bar{y}z$ (b) $E_2 = xy + z$

Figura 2.4: Mapas de Karnaugh de las expresiones booleanas dadas.

2.8.3. Caso de cuatro variables

El mapa de Karnaugh correspondiente a las expresiones booleanas $E = E(x, y, z, t)$ con cuatro variables x, y, z y t aparece en la Figura 2.5. En este caso habría dieciséis minitérminos posibles con cuatro literales:

$$xyzt, xyz\bar{t}, xy\bar{z}t, xy\bar{z}\bar{t}, ..., \bar{x}\bar{y}\bar{z}\bar{t}$$

están representados por los dieciséis cuadrados del mapa, como se observa en la figura.

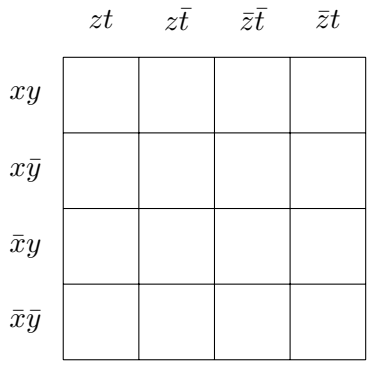


Figura 2.5: Mapa de Karnaugh de cuatro variables.

Un rectángulo básico en el mapa de Karnaugh con cuatro variables como un cuadrado, dos cuadrados adyacentes, cuatro cuadrados que forman un rectángulo de uno por cuatro o dos por dos, u ocho cuadrados que forman un rectángulo de dos por cuatro. Estos rectángulos básicos corresponden a productos fundamentales de cuatro, tres, dos y un literal, respectivamente. De nuevo, los rectángulos básicos maximales son los implicantes primos.

Ejemplo 20. Sea $E = x\bar{y} + xyz + \bar{x}\bar{y}\bar{z} + \bar{x}yz\bar{t}$, como se observa en la Figura 2.6 encontramos una forma de suma de productos minimal.

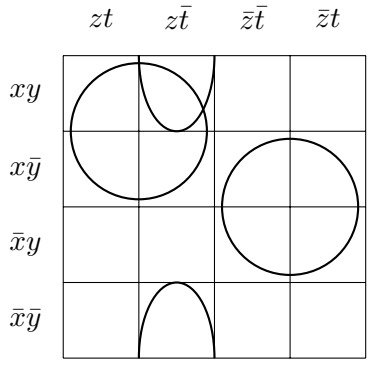


Figura 2.6: $E = xz + \bar{y}\bar{z} + yz\bar{t}$

2.9. Algoritmo de Quine-McCluskey

El algoritmo de Quine-McCluskey es un método que tiene que el mismo objetivo que los mapas de Karnaugh, encontrar implicantes primos y formas minimales para expresiones booleanas. La gran diferencia es que se puede aplicar este algoritmo a expresiones booleanas de más de 6 variables. En este trabajo se van a trabajar con hasta 8 variables, pero se podrían con muchas más.

Para la simplificación de una expresión booleana de una función booleana $f : \{0,1\}^n \to \{0,1\}$ mediante el manejo de primos implicantes

$$S(f) = \{b = b_1 b_2 ... b_n \in \{0,1\}^n : f(b) = 1\}$$

el proceso a seguir es el siguiente:

Algoritmo:	Se describen los minitérminos de la expresión dada por su equivalencia en números binarios.
Paso 1.	Se agrupan los elementos por bloques, en una columna, según el número de unos presentes en ese elemento, y ordenados en orden ascendente.
Paso 2.	Se compara cada elemento del bloque superior con todos los del siguiente bloque inferior y, en caso de que difieran en un único dígito se marcan ambos elementos. Se pasa a una segunda tabla el elemento que resulta de sustituir por un guión el dígito diferente, los dígitos restantes se mantienen iguales.
Paso 3.	Se repite el paso anterior con los bloques de la segunda tabla (se compara los elementos con los del bloque siguiente que tengan la misma posición "-") construyendo una tercera tabla, y así sucesivamente hasta que quede una tabla con un único bloque o no se puedan realizar más emparejamientos.
Paso 4.	Cuando no se pueda continuar con el proceso anterior, se crea una tabla de implicantes primos: ▪ Se consideran los implicantes primos como filas. ▪ Se consideran los términos dados por f como columnas.
Paso 5.	Se marcará una celda con una cruz si el término está cubierto por el implicante específico.
Paso 6.	Se seleccionan sólo aquellos implicantes que son necesarios para cubrir todos los términos, es decir, las celdas con una cruz, que pasan a ser implicantes primos esenciales.
Paso 7.	La expresión booleana formada por la suma de las expresiones correspondientes a los elementos seleccionados (implicantes primos esenciales) es una expresión simplificada.

Cuadro 2.5: Algoritmo de Quine-McCluskey

2.9.1. Ejemplo 1 del algoritmo de Quine-McCluskey

Sea la función

$$f(x, y, z, t) = \sum m(2, 5, 6, 7, 10, 13, 14, 15)$$

describimos los minitérminos de la expresión por su equivalencia en números binarios:

$$\{0010,\ 0101,\ 0110,\ 0111,\ 1010,\ 1101,\ 1110,\ 1111\}$$

Para simplicar esta función, aplicamos el método de Quine-McCluskey. Agrupamos los minitérminos por bloques, según el número de 1s:

m	x	y	z	t	Número de 1s
2	0	0	1	0	1
5	0	1	0	1	2
6	0	1	1	0	
10	1	0	1	0	
7	0	1	1	1	3
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	4

Ahora, comparamos cada elemento del bloque superior con todos los del siguiente bloque inferior y, en caso de que difieran en un único dígito se marcan ambos elementos. Se pasa a una segunda tabla el elemento que resulta de sustituir por un guión el dígito diferente, los dígitos restantes se mantienen iguales:

m	x	y	z	t	Número de 1s
(2,6)	0	-	1	0	1
(2,10)	-	0	1	0	
(5,7)	0	1	-	1	2
(5,13)	-	1	0	1	
(6,7)	0	1	1	-	
(6,14)	-	1	1	0	
(10,14)	1	-	1	0	
(7,15)	-	1	1	1	3
(13,15)	1	1	-	1	
(14,15)	1	1	1	-	

Volvemos a repetir el anterior paso, con la tabla que hemos construido:

m	x	y	z	t	Número de 1s
(2,6,10,14)	-	-	1	0	1
(5,7,13,15)	-	1	-	1	2
(6,7,14,15)	-	1	1	-	

Ahora se construye la tabla asociada con los implicantes primos:

	1010	1101	0101	0010	1111	1110	0111	0110
–10	X			X		X		X
-1-1		X	X		X		X	
-11-					X	X	X	X

En la tabla se observa que con los implicantes --10 y -1-1 se cubren todos los minitérminos. Por tanto, la expresión simplificada de f es: $f(x, y, z, t) = zt' + yt$.

2.9.2. Ejemplo 2 del algoritmo de Quine-McCluskey

Sea la función

$$f(x, y, z, t, w) = \sum m(0, 2, 3, 5, 7, 8, 10, 11, 13, 15, 22, 29, 30)$$

describimos los minitérminos de la expresión por su equivalencia en números binarios:

{00000, 00010, 00011, 00101, 00111, 01000, 01010, 01011, 01101, 01111, 10110, 11101, 11110}

Para simplicar esta función, aplicamos el método de Quine-McCluskey. Agrupamos los minitérminos por bloques, según el número de 1s:

m	x	y	z	t	w	Número de 1s
0	0	0	0	0	0	0
2	0	0	0	1	0	1
8	0	1	0	0	0	
3	0	0	0	1	1	2
5	0	0	1	0	1	
10	0	1	0	1	0	
7	0	0	1	1	1	3
11	0	1	0	1	1	
13	0	1	1	0	1	
22	1	0	1	1	0	
15	0	1	1	1	1	4
29	1	1	1	0	1	
30	1	1	1	1	0	

Ahora, comparamos cada elemento del bloque superior con todos los del siguiente bloque inferior y, en caso de que difieran en un único dígito se marcan ambos elementos. Se pasa a una segunda tabla el elemento que resulta de sustituir por un guión el dígito diferente, los dígitos restantes se mantienen iguales:

m	x	y	z	t	w	Número de 1s
(0,2)	0	0	0	-	0	0
(0,8)	0	-	0	0	0	
(2,3)	0	0	0	1	-	1
(2,10)	0	-	0	1	0	
(8,10)	0	1	0	-	0	
(3,7)	0	0	-	1	1	2
(3,11)	0	-	0	1	1	
(5,7)	0	0	1	-	1	
(5,13)	0	-	1	0	1	
(10,11)	0	1	0	1	-	
(7,15)	0	-	1	1	1	3
(11,15)	0	1	-	1	1	
(13,15)	0	1	1	-	1	
(13,29)	-	1	1	0	1	
(22,30)	1	-	1	1	0	

Volvemos a repetir el anterior paso, con la tabla que hemos construido:

m	x	y	z	t	w	Número de 1s
(0,2,8,10)	0	-	0	-	0	0
(2,3,10,11)	0	-	0	1	-	1
(3,7,11,15)	0	-	-	1	1	2
(5,7,13,15)	0	-	1	-	1	

Ahora se construye la tabla asociada con los implicantes primos:

	00000	00010	00011	00101	00111	01000	01010
0-0-0				X	X		
0-01-			X		X		
0--11		X	X				X
0-1-1	X	X				X	X
1-110							
-1101							

	01011	01101	01111	10110	11101	11110
0-0-0		X	X			
0-01-	X		X			
0--11	X					
0-1-1						
1-110				X		X
-1101		X			X	

En la tabla, podemos observar que hay dos soluciones para la selección de los primos implicantes esenciales:

1. 0--11, -1101, 1-110, 0-0-0 y 0-1-1. La función simplicada sería $f(x, y, z, t, w) = x'tw + yzt'w + xztw' + z'z'w' + x'zw$
2. 0-01-, -1101, 1-110, 0-0-0 y 0-1-1. La función simplicada sería $f(x, y, z, t, w) = x'z't + yzt'w + xztw' + z'z'w' + x'zw$

Capítulo 3

Código

En este capítulo se va a explicar las librerías que se han usado para programar los algoritmos en Python, además de mostrar el código y la explicación del funcionamiento de cada método para la resolución del algoritmo de Quine-McCluskey y mapas de Karnaugh explicados en el capítulo anterior.

3.1. Entorno de desarrollo

3.1.1. Visual Studio Code

Visual Studio Code (VS Code) [8] es un editor de código fuente gratuito, de código abierto y funciona en múltiples plataformas como Windows, macOS y Linux. Está desarrollado por Microsoft, una compañía con una dilatada experiencia en la creación de IDEs (entornos de desarrollo integrados). VS Code es capaz de adaptarse a cualquier lenguaje de programación, por lo que es muy conveniente para trabajar en distintos proyectos. Además, viene con un soperte integrado para el desarrollo de aplicaciones web. Algunas de sus características clave incluyen:

- La simplificación del desarrollo de frontend y backend es una de las grandes ventajas de Visual Studio Code. En el frontend, ofrece funcionalidades específicas para frameworks como Angular, mientras que en el backend, proporciona un soporte robusto para lenguajes como Python. Esta integración permite a los desarrolladores manejar ambos aspectos del desarrollo web de manera eficiente y coherente, todo desde un único entorno de desarrollo integrado.
- Proporciona una función de autocompletado inteligente, llamado IntelliSense, que incluye una serie de características que permiten obtener más información sobre el código que usa, realizar el seguimiento de los parámetros que escribe y agregar llamadas rápidamente a propiedades y a métodos. Pese a ser una herramienta de gran utilidad no se ha trabajado con ella para el desarrollo de este código.
- Herramientas de depuración integradas, que permite configurar puntos de interrupción, inspeccionar variables y ver la pila de llamadas.
- Integración nativa con Git, el sistema de control de versiones que usa ”Calculadora”, lo que permite que hagamos uso de las herramientas de este software

sin salirnos de VS Code y ayudados por sus interfaces gráficas. Con ello se hace muy sencillo trabajar con repositorios y realizar operaciones de commit, pull, push o resolver conflictos.

3.1.1.1. Python

Python [9] es un lenguaje de programación interpretado, de alto nivel y de propósito general, conocido por su legibilidad y simplicidad. Es especialmente adecuado para proyectos de algoritmos matemáticos debido a su sintaxis, y a la extensa biblioteca estándar que incluye numerosos módulos para matemáticas y ciencias de datos, como NumPy, SciPy y pandas. Estos módulos proporcionan herramientas poderosas para realizar cálculos complejos y manipulación de datos de manera eficiente.

La elección de Python para el backend del proyecto "Calculadora" permite aprovechar estas bibliotecas, facilitando el desarrollo y la implementación de algoritmos matemáticos avanzados. Aunque no tuve elección en la selección de Python, ya que era el lenguaje designado para este proyecto, su utilización ha demostrado ser muy beneficiosa para cumplir con los requisitos funcionales de los métodos implementados para la realización del algoritmo.

3.1.1.2. Angular

Angular [10] es un framework de desarrollo de aplicaciones web de código abierto, mantenido por Google. Está basado en TypeScript y proporciona una estructura robusta para la creación de aplicaciones. Angular es conocido por su capacidad para manejar vistas dinámicas, su arquitectura modular y su eficiente sistema de enlace de datos bidireccional, que permite una sincronización fluida entre el modelo y la vista.

Para el frontend del proyecto "Calculadora", Angular ha sido la opción elegida debido a sus potentes herramientas para el desarrollo de interfaces de usuario interactivas y dinámicas. La modularidad de Angular facilita la separación de preocupaciones, permitiendo que distintos componentes de la interfaz se desarrollen y mantengan de manera independiente. Como la elección de Python, tampoco tuve elección en la selección de Angular, ya que era el framework designado para este proyecto, pero su utilización ha permitido crear una interfaz de usuario eficiente y fácil de usar.

3.1.2. GitHub

Git es un sistema distribuido de control de versiones, gratuito y de código abierto bajo licencia GPLv2. Fue diseñado orginalmente por Linus Torvalds, creador del kernel de Linux. GitHub es un servicio de alojamiento en la nube de código fuente basado en el sistem de control de versiones que Git ofrece para manajar repositorios.

Este servicio nos permite tener un control de versiones, es decir, un sistema que registra cada cambio que se realiza en el código fuente del proyecto. Esto nos permite tener un histórico de todos los cambios producidos en sus ficheros, saber quién lo hizo y cuándo. Pero su importancia no sólo es que genera una copia de seguridad de los cambios, sino que también permite que distintas personas puedan colaborar en el desarrollo de una misma aplicación, facilitando la sincronización y versionado de sus cambios, así como la revisión de estos por otras personas.

Dado esto, GitHub es un servicio ideal para el proyecto "Calculadora", donde varios estudiantes implementan diferentes partes del código. La capacidad de integrar estas contribuciones de manera eficiente es fundamental, y GitHub facilita este proceso. Además, al trabajar en distintas ramas y disponer de copias de seguridad, se mejora la detección de fallos. En este proyecto, los componentes desarrollados se subieron a la rama principal (main) tras una prueba previa por parte de la persona encargada de los distintos repositorios, asegurando la calidad y estabilidad del código integrado. [11]

3.2. Librerías y módulos utilizados

3.2.1. Librerías de Python

3.2.1.1. itertools

itertools es una librería estándar de Python que proporciona herramientas para crear iteradores eficientes y complejos. Es especialmente útil para trabajar con combinaciones, permutaciones y productos cartesianos.

En este proyecto, itertools se utiliza principalmente mediante la función `product` para generar todas las combinaciones posibles de '0' y '1' que reemplazan los guiones en los minitérminos. También se usa la función `combinations` para generar combinaciones de elementos.

3.2.2. Módulos

3.2.2.1. exceptions.MyException

Este módulo interno define excepciones personalizadas que se adaptan a las necesidades específicas del proyecto. La clase `MyException` se utiliza para manejar errores específicos del proyecto de manera controlada y proporcionar mensajes de error personalizados que facilitan la depuración del código.

3.2.2.2. pasos.getPasos

Este módulo interno se utiliza para organizar y mostrar los pasos de las operaciones realizadas durante la ejecución del algoritmo de Quine-McCluskey y la generación de mapas de Karnaugh. La función `getPasos` se invoca para documentar y devolver los resultados intermedios y finales de las operaciones del algoritmo. Esto incluye la identificación de los implicantes primos esenciales y la creación de la tabla asociada.

3.3. Funciones auxiliares

En esta sección se explican las funciones auxiliares que son fundamentales para el funcionamiento del algoritmo Quine-McCluskey: `comparar`, `cubre`, `combinaciones`, `encontrar_todas_soluciones_dfs`, `encontrar_implicantes_primos_esenciales`, `tabla_asociada` e `inicializar_mapa_karnaugh`.

3.3.1. Función `comparar`

La función `comparar` toma como entrada dos minitérminos y verifica si se pueden combinar. Si los minitérminos difieren en exactamente una variable, la función combina los dos minitérminos, reemplazando la variable diferente con un guion ('-') y devuelve el resultado junto con un valor booleano `True`. Si los minitérminos difieren en más de una variable, devuelve el primer minitérmino (no tiene mucha importancia la devolución del minitérmino en este caso) y `False`.

Listing 3.1: Definición de la función `comparar`

```
def comparar(minitermino1, minitermino2):
    diferencias = 0
    resultado = ''
    for i in range(len(minitermino1)):
        if minitermino1[i] != minitermino2[i]:
            diferencias += 1
            resultado += '-'
        else:
            resultado += minitermino1[i]
    return (resultado, True) if diferencias == 1 else (minitermino1, False)
```

El funcionamiento es el siguiente:

- Se inicializan las variables `diferencias` (sirve para no comparar minitérminos que tengan más de una variable diferente) y `resultado`.
- Se itera sobre cada variable de los minitérminos. Si las variables son diferentes, se incrementa el contador de `diferencias` y se añade un guion al resultado. Si son iguales, se añade la variable al resultado.
- Si hay exactamente una diferencia, se retorna el resultado combinado y `True`. De lo contrario, se retorna el primer minitérmino y `False`.

3.3.2. Función `cubre`

La función `cubre` verifica si un implicante primo cubre un minitérmino dado. Compara cada variable del minitérmino con el implicante, y si encuentra una variable que no coincide y no es un guion, devuelve `False`. Si todas las variables coinciden o son guiones, devuelve `True`.

Listing 3.2: Definición de la función `cubre`

```
def cubre(minitermino, implicante):
    for m, p in zip(minitermino, implicante):
        if p != '-' and m != p:
            return False
    return True
```

El funcionamiento es el siguiente:

- Se itera sobre cada par de variables del minitérmino y el implicante utilizando `zip`.

- Si la variable del implicante no es un guion y no coincide con la variable del minitérmino, se retorna `False`.
- Si todas las variables son comparables (guiones o iguales), se retorna `True`.

3.3.3. Función `combinaciones`

La función `combinaciones` genera todas las combinaciones posibles de minitérminos reemplazando guiones por '0' y '1'. Esta función es útil para el caso en el que el input lleva guiones.

Listing 3.3: Definición de la función `combinaciones`

```
def combinaciones(input):
    miniterminos = input.split(',')
    resultado = []
    for minitermino in miniterminos:
        if '-' in minitermino:
            num_gaps = minitermino.count('-')
            remplazos = product('01', repeat=num_gaps)
            for remplazo in remplazos:
                new_minitermino = minitermino
                for r in remplazo:
                    new_minitermino = new_minitermino.replace('-', r, 1)
                resultado.append(new_minitermino)
        else:
            resultado.append(minitermino)
    return sorted(set(resultado), key=lambda x: (x.count('1'), int(x, 2)))
```

El funcionamiento es el siguiente:

- Se divide la cadena de entrada en minitérminos separados por comas.
- Para cada minitérmino, si contiene guiones, se cuentan los guiones y se generan todas las combinaciones posibles de '0' y '1' para esos guiones.
- Para cada combinación generada, se reemplazan los guiones en el minitérmino original.
- Se añaden todas las combinaciones generadas a la lista de resultados.
- Finalmente, se eliminan duplicados y se ordenan los resultados, retornando una lista de minitérminos ordenada.

3.3.4. Función `encontrar_todas_soluciones_dfs`

La función `encontrar_todas_soluciones_dfs` utiliza una búsqueda en profundidad (DFS) para encontrar todas las posibles combinaciones de implicantes primos que cubren todos los minitérminos.

Listing 3.4: Definición de la función `encontrar_todas_soluciones_dfs`

```
def encontrar_todas_soluciones_dfs(miniterminos, implicantes_primos,
    solucion, todas_soluciones, indice, tabla_cobertura):
    if indice == len(miniterminos):
```

```
3           solucion_ordenada = sorted(solucion)
4           if solucion_ordenada not in todas_soluciones:
5               todas_soluciones.append(solucion_ordenada)
6           return
7
8       minitermino = miniterminos[indice]
9       for implicante_primo in implicantes_primos:
10          if implicante_primo in tabla_cobertura[minitermino]:
11              nuevo_agregado = False
12              if implicante_primo not in solucion and set(tabla_cobertura[
                    minitermino]).isdisjoint(set(solucion)):
13                  solucion.append(implicante_primo)
14                  nuevo_agregado = True
15
16              encontrar_todas_soluciones_dfs(miniterminos, implicantes_primos
                    , solucion, todas_soluciones, indice + 1, tabla_cobertura)
17
18              if nuevo_agregado:
19                  solucion.remove(implicante_primo)
```

El funcionamiento es el siguiente:

- Si el índice actual es igual a la longitud de los minitérminos, se encontró una solución. La solución se ordena y se añade a la lista de todas las soluciones si no está ya presente.
- Para cada minitérmino, se itera sobre cada implicante primo. Si el implicante primo cubre el minitérmino actual y no está ya en la solución ni se solapa con la solución actual, se añade a la solución temporal.
- Se realiza una llamada recursiva al siguiente índice de minitérminos.
- Después de la llamada recursiva, se elimina el implicante primo de la solución si fue agregado en esta iteración.

3.3.5. Función `encontrar_implicantes_primos_esenciales`

La función `encontrar_implicantes_primos_esenciales` encuentra los implicantes primos esenciales dados los minitérminos e implicantes primos.

Listing 3.5: Definición de la función `encontrar_implicantes_primos_esenciales`

```
1  def encontrar_implicantes_primos_esenciales(cadena_miniterminos,
       cadena_implicantes_primos):
2       miniterminos = cadena_miniterminos.split(',')
3       implicantes_primos = cadena_implicantes_primos.split(',')
4
5       tabla_cobertura = {m: [] for m in miniterminos}
6       tabla_cobertura_imp = {p: [] for p in implicantes_primos}
7
8       for m in miniterminos:
9           for p in implicantes_primos:
10              if cubre(m, p):
11                  tabla_cobertura[m].append(p)
12                  tabla_cobertura_imp[p].append(m)
```

```
13
14      todas_soluciones = []
15      encontrar_todas_soluciones_dfs(miniterminos, implicantes_primos, [],
            todas_soluciones, 0, tabla_cobertura)
16
17      for solucion in todas_soluciones:
18          for i in range(0, len(solucion)):
19              for j in range(i + 1, len(solucion)):
20                  if tabla_cobertura_imp[solucion[i]] == tabla_cobertura_imp[
                        solucion[j]]:
21                      solucion.remove(solucion[j])
22
23      return [', '.join(solucion) for solucion in todas_soluciones]
```

El funcionamiento es el siguiente:

- Divide las cadenas de entrada en listas de minitérminos e implicantes primos.
- Crea tablas de cobertura para los minitérminos e implicantes primos.
- Llena las tablas de cobertura basándose en qué implicantes cubren cada minitérmino.
- Inicializa una lista de todas las soluciones y encuentra todas las soluciones posibles usando DFS.
- Elimina duplicados basados en la tabla de cobertura de implicantes.
- Convierte las soluciones a cadenas de implicantes separados por comas.

3.3.6. Función `tabla_asociada`

La función `tabla_asociada` crea una tabla que muestra qué implicantes primos cubren cada minitérmino.

Listing 3.6: Definición de la función `tabla_asociada`

```
1  def tabla_asociada(miniterminos_string, primos_implicantes_string):
2      miniterminos = miniterminos_string.split(',')
3      primos_implicantes = primos_implicantes_string.split(',')
4
5      tabla_asociada = [[''] * (len(miniterminos) + 1) for _ in range(len(
           primos_implicantes) + 1)]
6
7      for i, m in enumerate(miniterminos, start=1):
8          tabla_asociada[0][i] = m
9
10     for i, f in enumerate(primos_implicantes, start=1):
11         tabla_asociada[i][0] = f
12
13     for i, m in enumerate(miniterminos, start=1):
14         for j, f in enumerate(primos_implicantes, start=1):
15             if cubre(m, f):
16                 tabla_asociada[j][i] = 'X'
17
18     return tabla_asociada
```

El funcionamiento es el siguiente:

- Divide las cadenas de minitérminos e implicantes primos en listas.
- Inicializa la tabla asociada con una fila y columna adicionales para los encabezados.
- Llena la primera fila con los minitérminos como encabezados.
- Llena la primera columna con los implicantes primos como encabezados.
- Rellena la tabla con 'X' donde un implicante primo cubre un minitérmino.

3.4. algoritmo_Quine_McCluskey

La función `algoritmo_Quine_McCluskey` se utiliza para obtener los implicantes primos de un conjunto de minitérminos representados como una cadena de texto.

Listing 3.7: Definición de la función

```
def algoritmo_Quine_McCluskey(miniterminos_string):
    # Se llama a combinaciones para el caso en el que el input tenga guiones
    miniterminos = combinaciones(miniterminos_string)
    miniterminos_string = ','.join(miniterminos)

    combinado = True
    resultados = []
    cont = 1

    while combinado:
        implicantes = []

        # Creaci n de las tablas
        for minitermino in miniterminos:
            num_unos = minitermino.count('1') # N mero de unos
            binario = [int(bit) if bit in '01' else '-' for bit in minitermino] # Se separa el minit rmino en una lista de 0 y 1
            miniterminos_usados = [int(x, 2) for x in combinaciones(minitermino)] # Se guardan los minit rminos usados en decimal
            implicantes.append({
                "miniterminos": miniterminos_usados,
                "binario": binario,
                "num_unos": num_unos
            })
        resultados.append(implicantes)
        cont += 1

        next_step = []
        combinado = False
        usado = set()

        for i in range(len(miniterminos)):
            for j in range(i+1, len(miniterminos)):
```

```
32                  if(miniterminos[j].count('1') - miniterminos[i].count('1')
                        > 1):
33                      break
34                  result, puede_combinarse = comparar(miniterminos[i],
                        miniterminos[j])
35                  if puede_combinarse:
36                      combinado = True
37                      usado.add(miniterminos[i])
38                      usado.add(miniterminos[j])
39                      if result not in next_step:
40                          next_step.append(result)
41          miniterminos = [m for m in miniterminos if m not in usado] +
                next_step
42
43      prime_implicants_string = ','.join(miniterminos)
44
45      essential_primes = encontrar_implicantes_primos_esenciales(
            miniterminos_string, prime_implicants_string)
46      tabla = tabla_asociada(miniterminos_string, prime_implicants_string)
47
48      return pasos.algoritmo_Quine_McCluskey(resultados, essential_primes,
            tabla)
```

El funcionamiento de la función es el siguiente:

1. **Inicialización**: La función comienza llamando a la función `combinaciones` para generar todas las combinaciones posibles de minitérminos en caso de que el input contenga guiones. Los minitérminos resultantes se almacenan en una lista y se reconstruye la cadena de minitérminos.
2. **Bucle principal**: Se inicia un bucle que continuará mientras haya combinaciones posibles de minitérminos. Dentro de este bucle:
   - **Creación de tablas**: Se crean tablas para cada minitérmino, donde se cuenta el número de unos, se separan los bits en una lista y se almacenan los minitérminos utilizados en decimal.
   - **Comparación de minitérminos**: Se comparan los minitérminos entre sí para verificar si pueden combinarse. Si dos minitérminos pueden combinarse (difieren en solo un bit), se marca como combinado y se añade el resultado de la combinación a la lista para el siguiente paso.
3. **Actualización de minitérminos**: Al finalizar cada iteración del bucle, se actualiza la lista de minitérminos eliminando los que ya se han utilizado y añadiendo los resultados de las combinaciones posibles.
4. **Implicantes primos**: Una vez que no se pueden hacer más combinaciones, se generan los implicantes primos y se almacenan como una cadena de texto.
5. **Implicantes primos esenciales**: Para identificar los implicantes primos esenciales a partir de la cadena de minitérminos y los implicantes primos obtenidos se utiliza la función `encontrar_implicantes_primos_esenciales`.
6. **Tabla asociada**: Se crea una tabla asociada que muestra qué implicantes cubren cada minitérmino.

Finalmente, se llama a la función `pasos.algoritmo_Quine_McCluskey` para documentar y devolver los resultados intermedios y finales del algoritmo.

3.5. `mapa_Karnaugh`

La función `mapa_Karnaugh` es similiar a `algoritmo_Quine_McCluskey` pero se añade la creación de los mapas de Karnaugh.

Listing 3.8: Definición de la función

```
def mapa_Karnaugh(miniterminos_string):
    """
    Obtiene los implicantes primos de un string de minit rminos separados
       por comas.
    Devuelve un string de implicantes primos separados por comas.

    Args:
        cadena_miniterminos (str): Cadena de minit rminos separados por
           comas.

    Returns:
        list: Mapa de Karnaugh con implicantes primos esenciales y tabla
           asociada.
    """

    # Generar todas las combinaciones posibles de minit rminos
    miniterminos = combinaciones(miniterminos_string)
    miniterminos_string = ','.join(miniterminos)

    # Inicializar variables
    combinado = True
    resultados = []
    contador = 1
    num_variables = len(miniterminos[0])
    mapas_karnaugh = []
    mapa_karnaugh = inicializar_mapa_karnaugh(num_variables)

    # Ciclo para iterar mientras se pueda combinar
    while combinado:
        implicantes = []

        # Crear la lista de implicantes con informaci n detallada
        for cadena_binaria in miniterminos:
            num_unos = cadena_binaria.count('1')
            lista_binaria = [int(bit) if bit in '01' else '-' for bit in
               cadena_binaria]
            miniterminos_usados = [int(x, 2) for x in combinaciones(
               cadena_binaria)]
            implicantes.append({
                "miniterminos": miniterminos_usados,
                "binario": lista_binaria,
                "num_unos": num_unos
            })

            # Llenar el mapa de Karnaugh
```

```
41              for i in range(1, len(mapa_karnaugh[0])):
42                  for j in range(1, len(mapa_karnaugh)):
43                      resultado = cubre(mapa_karnaugh[0][i] + mapa_karnaugh[j
                            ][0], cadena_binaria)
44
45                      if resultado:
46                          if contador == 1:
47                              mapa_karnaugh[j][i] = f'{mapa_karnaugh[0][i] +
                                    mapa_karnaugh[j][0]}'
48                          elif mapa_karnaugh[j][i] != '0':
49                              mapa_karnaugh[j][i] = f'{mapa_karnaugh[0][i] +
                                    mapa_karnaugh[j][0]}'
50
51          # Copiar el mapa de Karnaugh actual para la lista de mapas
52          mapa_karnaugh_aux = copy.deepcopy(mapa_karnaugh)
53          mapas_karnaugh.append(mapa_karnaugh_aux)
54          resultados.append(implicantes)
55          contador += 1
56
57          # Combinar minit rminos
58          siguiente_paso = []
59          combinado = False
60          usados = set()
61          for i in range(len(miniterminos)):
62              for j in range(i + 1, len(miniterminos)):
63                  if (miniterminos[j].count('1') - miniterminos[i].count('1')
                        > 1):
64                      break
65                  resultado, puede_combininarse = comparar(miniterminos[i],
                        miniterminos[j])
66                  if puede_combininarse:
67                      combinado = True
68                      usados.add(miniterminos[i])
69                      usados.add(miniterminos[j])
70                      if resultado not in siguiente_paso:
71                          siguiente_paso.append(resultado)
72          miniterminos = [m for m in miniterminos if m not in usados] +
                siguiente_paso
73
74
75      primos_implicantes_string = ','.join(miniterminos)
76      essential_primes = encontrar_implicantes_primos_esenciales(
            miniterminos_string, primos_implicantes_string)
77      tabla = tabla_asociada(miniterminos_string, primos_implicantes_string)
78
79      return pasos.mapa_Karnaugh(resultados, essential_primes, tabla,
            mapas_karnaugh)
```

El funcionamiento de la función es el siguiente:

1. **Generación de combinaciones**: La función comienza llamando a la función `combinaciones` para generar todas las combinaciones posibles de minitérminos en caso de que el input contenga guiones. Los minitérminos resultantes se almacenan en una lista y se reconstruye la cadena de minitérminos.

2. **Inicialización**: Se inicializan las variables necesarias, incluyendo una lista para almacenar los resultados intermedios, un contador, el número de variables y el mapa de Karnaugh.
3. **Bucle principal**: Se inicia un bucle que continuará mientras haya combinaciones posibles de minitérminos. Dentro de este bucle:
   - **Creación de la lista de implicantes**: Se crea una lista de implicantes, donde se cuenta el número de unos, se separan los bits en una lista y se almacenan los minitérminos utilizados en decimal.
   - **Llenado del mapa de Karnaugh**: Se llena el mapa de Karnaugh comparando cada celda del mapa con la cadena binaria de los minitérminos. Si hay coincidencia, se actualiza la celda correspondiente.
   - **Copia del mapa de Karnaugh**: Se realiza una copia del mapa de Karnaugh actual y se añade a la lista de mapas de Karnaugh.
4. **Combinación de minitérminos**: Al finalizar cada iteración del bucle, se combinan los minitérminos verificando si se pueden combinar (difieren en solo un bit). Si dos minitérminos pueden combinarse, se marca como combinado y se añade el resultado de la combinación a la lista para el siguiente paso.
5. **Implicantes primos**: Una vez que no se pueden hacer más combinaciones, se generan los implicantes primos y se almacenan como una cadena de texto.
6. **Implicantes primos esenciales**: Para identificar los implicantes primos esenciales a partir de la cadena de minitérminos y los implicantes primos obtenidos se utiliza la función `encontrar_implicantes_primos_esenciales`.
7. **Tabla asociada**: Se crea una tabla asociada que muestra qué implicantes cubren cada minitérmino.

Finalmente, se llama a la función `pasos.mapa_Karnaugh` para documentar y devolver los resultados intermedios y finales del algoritmo junto con el mapa de Karnaugh generado en cada iteración.

Capítulo 4

Guía de uso

4.1. Casos de uso

En esta sección se describen los principales casos de uso de la aplicación basada en el algoritmo de Quine-McCluskey. Cada caso de uso está diseñado para guiar al usuario a través de las funcionalidades disponibles y facilitar su comprensión y manejo de la herramienta.

4.1.1. Caso de uso 1: Simplificación de Funciones Booleanas

- **ID:** CU-01
- **Actor:** Estudiante
- **Descripción:** Simplificación de funciones booleanas utilizando el algoritmo de Quine-McCluskey.
- **Pasos a Seguir:**
  1. El estudiante accede a la aplicación y selecciona la opción ”Algoritmo de Quine-McCluskey”.
  2. Introduce los minitérminos de la función booleana en el formato correcto (secuencias de ‘0‘, ‘1‘, y ‘-‘ separadas por comas).
  3. Hace click en el botón ”Calcular”.
  4. La aplicación muestra la solución simplificada, incluyendo los implicantes primos esenciales. También al darle click al botón ”Mostrar Pasos” se muestran los pasos intermedios.
  5. El estudiante puede revisar los pasos detallados y compararlos con su propio trabajo.

4.1.2. Caso de uso 2: Comparación con Mapas de Karnaugh

- **ID:** CU-02
- **Actor:** Estudiante

- **Descripción:** Comparar la solución del algoritmo de Quine-McCluskey con los mapas de Karnaugh.
- **Pasos a Seguir:**
  1. El estudiante accede a la aplicación y selecciona la opción "Mapas de Karnaugh".
  2. Introduce los minitérminos de la función booleana en el formato correcto (secuencias de '0', '1', y '-' separadas por comas).
  3. Hace click en el botón "Calcular".
  4. La aplicación muestra la solución simplificada, incluyendo los implicantes primos esenciales.
  5. Al darle click al botón "Mostrar Pasos"la aplicación muestra la solución simplificada utilizando ambos métodos: el algoritmo de Quine-McCluskey y los Mapas de Karnaugh. El estudiante puede ver la relación entre los términos de ambos métodos, resaltada al pasar el cursor sobre las celdas correspondientes.

4.1.3. Caso de uso 3: Uso de Ejemplos Predeterminados

- **ID:** CU-03
- **Actor:** Estudiante
- **Descripción:** Utilizar ejemplos predeterminados para aprender el uso de la aplicación.
- **Pasos a Seguir:**
  1. El usuario accede a la aplicación y selecciona una de las opciones disponibles (Algoritmo de Quine-McCluskey o Mapas de Karnaugh).
  2. En la pantalla principal, selecciona uno de los ejemplos predeterminados disponibles.
  3. La aplicación carga los datos del ejemplo en el input y muestra la solución al darle click en el botón "Calcular".
  4. El usuario puede revisar los pasos de la solución.

4.2. Interfaz

La interfaz gráfica se ha hecho con Angular, como se ha comentado previamente, utilizando HTML, CSS y TypeScript. En cuanto al diseño de todas las partes de la página, muchos componentes ya estaban creados por el proyecto "Calculadora", como el header, footer, botones, etc, solo hacía falta añadirlos.

En la Figura 4.1 se puede ver la vista de la pantalla donde está integrado el Algoritmo de Quine-McCluskey. Esta pantalla incluye una pequeña descripción para que el usuario sepa con qué tipo de algoritmo está trabajando. Además, se le permite seleccionar dos funcionalidades que se han desarrollado:

Figura 4.1: Vista de la pantalla que integra el Algoritmo de Quine-McCluskey.

1. **Algoritmo de Quine-McCluskey**: Al seleccionar esta opción, el usuario introducirá los minitérminos de su ejercicio y se mostrará directamente la solución, permitiendo al usuario ver las tablas del algoritmo de Quine-McCluskey y los pasos de la solución.
2. **Mapas de Karnaugh**: En esta opción, el usuario verá la misma solución que en la opción del Algoritmo de Quine-McCluskey, pero se mostrarán los pasos haciendo una comparativa con los mapas de Karnaugh y las tablas del algoritmo de Quine-McCluskey.

Dentro del input donde los usuarios pueden introducir los minitérminos con los que quieren trabajar, se permite al usuario añadir '0', '1' y '-', además de las comas que se utilizan para separar los minitérminos introducidos. Los guiones se han habilitado para estudiantes que estén en una tabla intermedia, por ejemplo, o para enunciados que tengan minitérminos con un gran número de variables. Cualquier otro input distinto a lo solicitado se manejará como un error, enviando un mensaje dependiendo del tipo de error. En este caso, los errores por inputs serían:

- *Error: No se proporcionó ningún parámetro de entrada.*
- *Parámetros inválidos: El número de variables tiene que ser entre 2 y 8 (para el Algoritmo de Quine-McCluskey), o entre 2, 3 o 4 (para los Mapas de Karnaugh).*
- *Parámetros inválidos: Sólo se permiten secuencias de '-', '0' y '1' separadas por comas.*
- *Parámetros inválidos: Todos los términos deben tener la misma cantidad de variables.*

Además, como se observa en la Figura 4.1 se permite a los usuarios usar distintos ejemplos que vienen predeterminados, para que en un primer momento puedan ver rápidamente el funcionamiento sin tener que buscar un ejemplo o un ejercicio para probarlo.

4.2.1. Solución del Algoritmo de Quine-McCluskey

Cuando el usuario introduce su ejercicio y hace clic en el botón de calcular, se muestra directamente la solución con los primos implicantes esenciales resultantes, como se puede ver en la Figura 4.2.

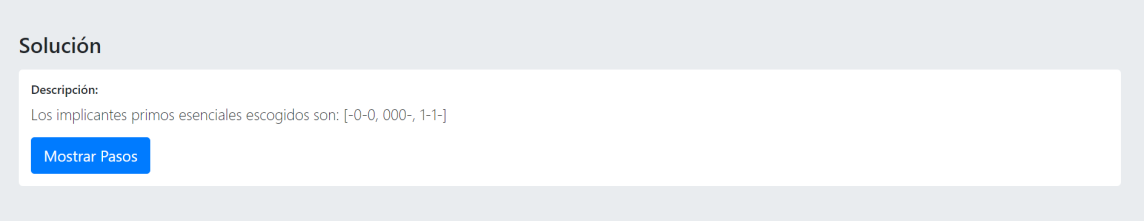


Figura 4.2: Vista de la solución con los primos implicantes esenciales seleccionados para el ejercicio.

Además, en la Figura 4.3, se puede observar que también se consideran ejercicios con múltiples soluciones posibles.

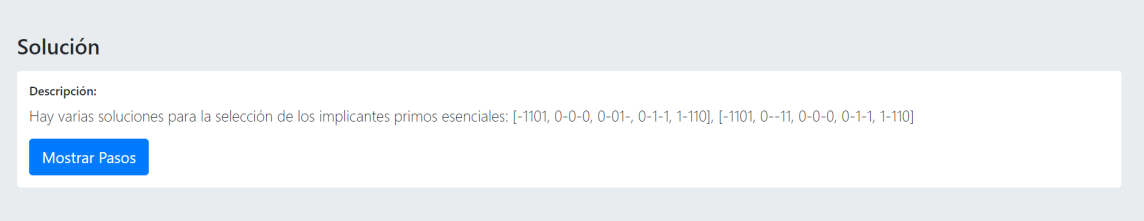


Figura 4.3: Ejemplo de una solución con múltiples resultados posibles.

Se permite la visualización de los pasos necesarios para llegar a esa solución. En la Figura 4.4, se muestra un ejemplo de una tabla que forma parte del algoritmo de Quine-McCluskey, donde se presentan los minitérminos en decimal, las variables y el número de unos. Para mostrar estos pasos, se ha seguido el procedimiento que utilizan los profesores en la asignatura de Matemática Discreta I en la Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid, con el fin de que sea familiar para los estudiantes.

Finalmente, se muestra la tabla intermedia con la solución final que incluye los primos implicantes esenciales, como se aprecia en la Figura 4.5.

4.2.2. Solución de Mapas de Karnaugh

La solución utilizando mapas de Karnaugh sigue un procedimiento muy similar al del algoritmo de Quine-McCluskey. En esta opción, se muestra tanto el procedimiento del algoritmo como el de los mapas, estableciendo una correlación entre ambos.

En la Figura 4.6, se puede apreciar que al poner el cursor sobre una fila de la tabla del Algoritmo de Quine-McCluskey, se resalta la correlación con los mapas de Karnaugh. Esto permite que el estudiante vea de manera más gráfica cómo se relacionan los términos del algoritmo con los mapas. De manera similar, al poner el cursor en una celda que contiene un 1 en los mapas de Karnaugh, se muestra su relación con la tabla del algoritmo de Quine-McCluskey.

Pasos

Paso 1:

La tabla es

Minitérminos	x_1	x_2	x_3	x_4	Número de Unos
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
8	1	0	0	0	1
10	1	0	1	0	2
11	1	0	1	1	3
14	1	1	1	0	3
15	1	1	1	1	4

Figura 4.4: Ejemplo de tabla del algoritmo de Quine-McCluskey, mostrando los minitérminos en decimal, las variables y el número de unos.

Paso 4:

La tabla asociada con los implicantes primos es

	0000	0001	0010	1000	1010	1011	1110	1111
000-	X	X						
-0-0	X		X	X	X			
1-1-					X	X	X	X

Paso 5:

Los implicantes primos esenciales escogidos son: [-0-0, 000-, 1-1-]

Figura 4.5: Tabla intermedia del algoritmo de Quine-McCluskey, mostrando la solución final con los primos implicantes esenciales.

Esta funcionalidad interactiva es particularmente útil para los estudiantes, ya que facilita la comprensión de ambos métodos de simplificación lógica y cómo se relacionan entre sí. Al poder visualizar directamente la correlación, los estudiantes pueden entender mejor el proceso de simplificación y los pasos intermedios que llevan a la solución final.

Paso 1:

La tabla es

Minitérminos	x_1	x_2	x_3	x_4	Número de Unos
1	0	0	0	1	1
4	0	1	0	0	1
3	0	0	1	1	2
5	0	1	0	1	2
9	1	0	0	1	2
12	1	1	0	0	2
11	1	0	1	1	3
13	1	1	0	1	3
14	1	1	1	0	3
15	1	1	1	1	4

	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	1	0	1	1
10	0	0	1	0

Figura 4.6: Visualización de la relación entre la tabla del Algoritmo de Quine-McCluskey y los Mapas de Karnaugh.

Capítulo 5

Impacto del trabajo

5.1. Impacto general

El desarrollo de una librería basada en el algoritmo de Quine-McCluskey tiene un significativo impacto en el ámbito educativo, además es el objetivo del proyecto. Al proporcionar una herramienta interactiva y autónoma para la simplificación de funciones booleanas, se facilita el aprendizaje independiente y el acceso a recursos educativos intersantes. Esta herramienta permite a los estudiantes practicar y mejorar sus habilidades sin la necesidad de supervisión constante, promoviendo así la autonomía y el aprendizaje continuo. Además, la resolución de los ejercicios mostando los pasos asegura que los usuarios comprendan los fundamentos del algoritmo y puedan aplicarlo efectivamente en sus estudios.

El proyecto tiene un impacto positivo en la investigación y el desarrollo (I+D), al proporcionar una herramienta accesible y eficiente para la simplificación de funciones booleanas, se ayuda al trabajo de investigadores y desarrolladores que necesitan este tipo de simplificaciones en sus proyectos. La librería puede integrarse en diversas plataformas de software, apoyando el desarrollo de nuevas tecnologías y productos. Además, el uso de Python como lenguaje de programación asegura que la herramienta sea fácilmente adoptada y adaptada por diversas comunidades de desarrolladores y usuarios, promoviendo la colaboración y el intercambio de conocimientos.

5.2. Objetivos de Desarrollo Sostenible

La Agenda 2030 es un plan de acción global y de colaboración cuyo objetivo principal es abordar los desafíos urgentes y complejos que enfrenta nuestro mundo. Esta agenda promueve el desarrollo sostenible en sus tres dimensiones: social, económica y ambiental, buscando permitir a las generaciones presentes y futuras vivir en un mundo justo, próspero y en equilibrio con el planeta.

La Agenda 2030 se compone de 17 Objetivos de Desarrollo Sostenible (ODS), que se basan en los principios de inclusión, igualdad y sostenibilidad. Para ver todos los Objetivos de Desarrollo Sostenible, véase [12].

Este Trabajo Fin de Grado está relacionado con los Objetivos de Desarrollo Sostenible que se enumeran en el resto del capítulo:

5.2.1. ODS 4: Educación de Calidad

Este proyecto está diseñado para asegurar una educación inclusiva, equitativa y de calidad, al tiempo que promueve oportunidades de aprendizaje continuas para todos. Facilita el acceso a la educación mediante una librería interactiva y autónoma que trabaja con el algoritmo de Quine-McCluskey, permitiendo a los estudiantes practicar y aprender de forma independiente, sin necesidad de la intervención constante de un docente. Además, se fomenta la calidad educativa proporcionando el marco teórico necesario para el uso de la librería.

Teniendo en cuenta los bajos niveles de habilidades en tecnologías de la información y las comunicaciones (TIC), el uso de una aplicación digital para el aprendizaje no solo enseña contenidos específicos, sino que también mejora las habilidades en este tipo de tecnologías. Y con estas herramientas de apoyo al estudiante, se ofrece una forma más dinámica y atractiva de aprender, ayudando a que puedan estar más motivados para continuar con su educación.

5.2.2. ODS 8: Trabajo Decente y Crecimiento Económico

Al promover la innovación tecnológica y el uso de herramientas digitales en la educación, se mejoran los métodos para la simplicación de funciones booleanas, aumentando la productividad al simplificar y automatizar procesos matemáticos complejos. Este enfoque también fomenta el espíritu emprendedor, permitiendo a otros utilizar la librería y generar valor económico a partir de su aplicación.

5.2.3. ODS 9: Industria, Innovación e Infraestructuras

Desarrollando una librería en Python, se aporta al fortalecimiento de la infraestructura digital en el ámbito educativo. Además, este proyecto demuestra cómo la tecnología y la programación pueden mejorar los procesos educativos y apoyar el aprendizaje autónomo, promoviendo así la innovación en el sector educativo.

Además, el proyecto fomenta la innovación en el sector tecnológico al poner a disposición del público las componentes desarrolladas. Esto permite a otros profesionales y desarrolladores utilizar y ampliar estas herramientas en sus propios proyectos, facilitando la colaboración y el avance tecnológico en la comunidad educativa y más allá. Al proporcionar una base de código abierta y documentada, se potencia la creación de nuevas aplicaciones y soluciones que pueden ser adaptadas a diferentes contextos y necesidades, impulsando así el desarrollo de infraestructuras tecnológicas robustas y sostenibles.

5.2.4. ODS 10: Reducción de Desigualdades

El objetivo de este proyecto es reducir las desigualdades tanto dentro de los países como entre ellos. Al proporcionar una herramienta digital accesible, se contribuye a disminuir la brecha digital en el acceso a la educación. La herramienta permite superar barreras como la ubicación geográfica o las limitaciones de tiempo, facilitando que más personas puedan participar en la educación matemática independientemente de sus circunstancias personales.

5.2.5. ODS 12: Producción y Consumo Responsables

El proyecto también busca asegurar patrones de producción y consumo sostenibles. Al proporcionar métodos eficientes y automatizados para la resolución de ecuaciones, se optimiza el proceso de aprendizaje y se reduce la necesidad de recursos adicionales como papel o cálculos manuales en formato impreso, contribuyendo así a la reducción del impacto ambiental.

La librería, al ser reutilizable y aplicable en múltiples contextos, fomenta la reutilización de herramientas y recursos, evitando la necesidad de crear soluciones individuales para cada problema específico.

Capítulo 6

Resultados y conclusiones

En este capítulo, se presentan los resultados obtenidos a lo largo del desarrollo del Trabajo de Fin de Grado (TFG), se discuten las conclusiones personales del autor y se proponen posibles direcciones para trabajos futuros. Este capítulo se divide en las siguientes secciones:

- **Resultados**: Se resumen los logros y hallazgos clave del TFG, destacando cómo la aplicación desarrollada contribuye al proyecto Çalculadoraz su impacto en el aprendizaje de los estudiantes.
- **Propuestas de aplicaciones en aula**: Se analiza cómo la aplicación puede ser utilizada en un entorno educativo para beneficiar tanto a estudiantes como a profesores, mejorando la dinámica de enseñanza y aprendizaje.
- **Conclusiones personales**: El autor reflexiona sobre la experiencia de desarrollo del proyecto, las habilidades adquiridas y el valor del proyecto para su crecimiento profesional.
- **Trabajo futuro**: Se identifican áreas potenciales de mejora y expansión para la aplicación, ofreciendo ideas para futuros proyectos y desarrollos.

6.1. Resultados

A modo resumen, este Trabajo de Fin de Grado forma parte de ”Calculadora”, un proyecto que ha creado una aplicación didáctica gracias a los Trabajos de Fin de Grado de los estudiantes de la Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid.

Esta aplicación será de gran ayuda para los estudiantes, ya que es una ampliación de los conocimientos impartidos en clase, proporciona un marco teórico y una serie de métodos que les permitirá profundizar en los distintos campos, en este caso, el campo de la simplifación booleana con el algoritmo de Quine-McCluskey y mapas de Karnaugh.

Como se ha visto, se proporciona una interfaz donde los estudiantes pueden seleccionar el método que quieran para la resolución, la diferencia está en la explicación de los pasos. El estudiante introducirá el enunciado del ejercicio en el que quiere trabajar o visualizar la solución, y les mostrará los primos implicantes esenciales, con el

procedimiento para llegar hasta ahí. Se pueden trabajar con enunciados que tengan varias soluciones posibles, está contemplado en la integración del algoritmo.

6.2. Propuestas de aplicaciones en aula

Desde su concepción, esta aplicación ha sido desarrollada para servir como una herramienta interactiva de apoyo para los estudiantes, con el objetivo de reforzar los conceptos teóricos impartidos en las clases.

El grupo beneficiado principal son los estudiantes. La aplicación les proporciona una plataforma web fiable con la capacidad de resolver ejercicios de las asignaturas que se imparten en la universidad, validada por los propios profesores. Esto les permitirá ahorrar tiempo al ofrecer una herramienta flexible que muestra las soluciones con todos los pasos detallados. Así, si un estudiante no tiene la solución del ejercicio que está realizando, puede utilizar la aplicación como un validador para confirmar si su respuesta es correcta. Además, la aplicación incentiva la autonomía del estudiante al permitirle trabajar a su propio ritmo y resolver ejercicios de manera independiente. Esto fomenta un aprendizaje más activo y autodirigido, donde los estudiantes pueden identificar y corregir sus errores sin depender constantemente del profesor. Asimismo, la aplicación promueve la innovación en el ámbito universitario al integrar herramientas tecnológicas avanzadas en el proceso de enseñanza y aprendizaje, lo cual es fundamental para preparar a los estudiantes para un entorno laboral cada vez más digitalizado.

Contribuye también a la reducción de la brecha digital al proporcionar acceso a recursos educativos de alta calidad a todos los estudiantes, independientemente de su nivel de habilidad tecnológica o recursos disponibles. La accesibilidad y facilidad de uso de la aplicación aseguran que tanto estudiantes con experiencia en tecnología como aquellos con menos habilidades digitales puedan beneficiarse de sus funcionalidades. De este modo, se garantiza una educación más equitativa y accesible para todos.

Los profesores también se verán beneficiados de esta aplicación. Podrán optimizar el tiempo invertido en las clases, enfocándose más en la enseñanza de la teoría y proporcionando ejercicios a los estudiantes para que los resuelvan en casa. En caso de que los estudiantes no sepan cómo resolver los ejercicios o deseen profundizar más en el tema, siempre podrán recurrir a esta herramienta.

Además, la aplicación puede ser utilizada para la evaluación continua. Los profesores pueden asignar ejercicios específicos y utilizar la herramienta para verificar las soluciones de los estudiantes, facilitando así la corrección y el seguimiento del progreso de cada alumno.

6.3. Conclusiones personales

Desde la parte teórica, ha sido un refuerzo de los conocimientos que he adquirido en mi carrera, centrándome en Matemática Discreta I. La investigación de conceptos como álgebra de Boole, dualidad, expresiones booleanas,... ha sido esencial para consolidar mi compresión en estos temas. Además, este trabajo me ha permitido desarrollar un enfoque crítico y métodico para buscar fuentes para buscar fuentes,

revisar anotaciones y entender diversos puntos de vista, lo que ha sido fundamental para la escritura de este marco teórico.

Este proyecto ha sido una experiencia muy enriquecedora, tanto en términos técnicos como personales. Trabajar con tecnologías como Angular para el frontend y Python para el backend me ha permitido mejorar mis habilidades en el desarrollo de aplicaciones web. Además, la integración de GitHub para el control de versiones ha facilitado enormemente la colaboración y el manejo del código, asegurando que todas las contribuciones se integren de manera eficiente.

He aprendido la importancia de una buena planificación y diseño del software, especialmente en proyectos colaborativos donde múltiples personas contribuyen con diferentes partes del código. La experiencia de seguir el proceso de desarrollo desde la concepción hasta la implementación y pruebas finales ha sido invaluable.

Asimismo, el hecho de que el proyecto forme parte de una herramienta didáctica que será utilizada por otros estudiantes añade un componente de responsabilidad y motivación adicional. Saber que mi trabajo ayudará a otros a comprender mejor conceptos complejos es muy gratificante.

6.4. Trabajo futuro

Existen varias áreas en las que se podría expandir y mejorar el trabajo realizado en este TFG:

- **Optimización del rendimiento**: Mejorar la eficiencia del algoritmo de Quine-McCluskey para manejar casos más complejos y con un mayor número de variables. Aunque el número de variables con el que se trabaja en este proyecto es suficiente para que los estudiantes puedan realizar sus ejercicios, una ampliación permitiría satisfacer la curiosidad de los propios alumnos interesados en casos más avanzados. Sin embargo, es importante señalar que si se implementa esta ampliación de variables, no sería posible visualizar todos los pasos intermedios debido a la complejidad. Este caso se menciona como un posible trabajo futuro, aunque no se considera de alta prioridad.
- **Mejoras en la interfaz de usuario**: Añadir más características interactivas y visualizaciones que ayuden a los estudiantes a comprender mejor los pasos del proceso de simplificación. Esto incluye ofrecer más información contextual y ayudas adicionales mediante botones o pop-ups que expliquen cada paso y aclaren conceptos que los estudiantes podrían no entender completamente. Estas mejoras harían la aplicación más amigable y educativa.

  Dentro de estas mejoras, se podría trabajar en la vista móvil, ya que actualmente se ha deshabilitado para una tamaño de pantalla determinado. Esto es debido a que no se veían bien las tablas, pese a intentarlo haciendo un scroll pero era muy tedioso y no valía la pena. Se propone en proyectos futuros ver una solución posible para este desafío.
- **Ejemplos preparados**: Aunque ya existen varios ejemplos predeterminados en la aplicación, se podría crear un banco de ejercicios más extenso que aparezcan aleatoriamente para los estudiantes. Esto les proporcionaría una variedad de ejercicios con los que practicar sin necesidad de tener un enunciado específico

o buscar ejercicios adicionales por su cuenta. Un banco de ejemplos robusto enriquecería la experiencia de aprendizaje y práctica.

- **Uso de librerías de Python**: Aunque no fue necesario para la realización de este trabajo, la integración de librerías de Python específicas podría ampliar las capacidades de la aplicación. Librerías adicionales podrían mejorar la eficiencia y funcionalidad del backend, proporcionando herramientas avanzadas para el procesamiento de datos y la ejecución de algoritmos complejos.

Estas mejoras no solo incrementarían la utilidad de la aplicación, sino que también ofrecerían a futuros estudiantes la oportunidad de contribuir al proyecto, aprendiendo y aplicando nuevas tecnologías y metodologías en el proceso.

Bibliografía

[1] Miguel Reyes Águeda Mata. *Matemática Discreta I (Guía de clase 2ª edición)*. UPM, 2013.

[2] Kenneth H. Rosen. *Matemática discreta y sus aplicaciones (5ª ED.)* McGraw-Hill, 2004.

[3] Elliott Mendelson. *Boolean Algebra and Switching Circuits*. McGraw-Hill, 2019.

[4] Sourangsu Banerji. *Computer Simulation Codes for the Quine-McCluskey Method of Logic Minimization*. Department of Electronics & Communication Engineering, RCC-Institute of Information Technology.

[5] Seymour Lipschutz. *Matemáticas discretas*. McGraw-Hill, 2009.

[6] Willard Quine. «The Problem of Simplifying Truth Functions». En: *The American Mathematical Monthly* 59:8 (1952), págs. 521-531.

[7] John Chu. «Some Methods for Simplifying Switching Circuits Using "Don't Care" Conditions». En: *Association for Computing Machinery* 8.4 (1961), págs. 497-512.

[8] Microsoft. *Visual Studio Code Documentation*. URL: `https://code.visualstudio.com/docs`.

[9] Python Software Foundation. *Python Documentation*. URL: `https://www.python.org/doc/`.

[10] Google. *Angular Documentation*. URL: `https://angular.io/docs`.

[11] Miguel Ángel Durán. *Aprendiendo Git y GitHub: Desde cero hasta buenas prácticas y estrategias de trabajo en equipo*.

[12] Naciones Unidas. *Objetivos de Desarrollo Sostenible*. 2015. URL: `https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/`.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Mon Jun 03 15:36:04 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)