Universidad
Europea
del Atlántico

LÓGICA

© TODOS LOS DERECHOS RESERVADOS

© TODOS LOS DERECHOS RESERVADOS

Universidad Europea del Atlántico

Índice

Presentación

1. Lógica proposicional

1.1. Introducción 3
1.2. Breve historia de la evolución de la lógica como ciencia 4
1.3. Introducción a la Lógica proposicional 5
1.4. Sintaxis de la lógica proposicional 10
1.4.1. Alfabeto o vocabulario 10
1.4.2. Gramática. Fórmulas bien formadas 13
1.5. Semántica de la Lógica proposicional 16
1.5.1. Tablas de verdad 16
1.5.2. Traducción de expresiones del lenguaje natural 27
1.5.2.1. Especificaciones de sistema 29
1.5.3. Interpretación de fórmulas 30
1.5.4. Clasificación de fórmulas 34
1.6. Diagrama de Decisión Binario (DDB) 36
1.7. Búsquedas booleanas 40
1.8. Juegos de Lógica 41
1.9. Lógica y operaciones con bits 43
1.10. Implicación y equivalencia lógica. Leyes de la lógica proposicional 44

2. Circuitos lógicos

2.1. Introducción 53
2.2. Compuertas lógicas. Diseño de circuitos lógicos 53
2.3. Formas normales 58
2.4. Algebra booleana 62
2.5. Formas canónicas 63

© Todos los derechos reservados

2.6. Simplificación de circuitos 67
2.6.1. Simplificación algebraica 68
2.6.2. Mapas de Karnaugh 70

3. Lógica de predicados

3.1. Introducción 81
3.2. Predicados 83
3.3. Cuantificadores 85
3.3.1. Cuantificador universal 86
3.3.2. Cuantificador existencial 88
3.3.3. Variables libres y ligadas 89
3.3.4. Negaciones 91
3.4. Sintaxis de la Lógica de predicado 92
3.4.1. Alfabeto o vocabulario de la lógica de predicados 92
3.4.2. Gramática. Términos y fórmulas 93
3.5. Formalización de expresiones del lenguaje natural en lógica de predicados 96
3.5.1. Cuantificadores anidados 98
3.6. Semántica de la lógica de predicado 103
3.6.1. Interpretación de fórmulas 103
3.7. Leyes de la Lógica predicados 107

4. Estructuras deductivas

4.1. Introducción 111
4.2. Estructuras deductivas 112
4.2.1. Esquema de las estructuras deductivas 113
4.2.2. Validez de las deducciones 114
4.3. Reglas de inferencia 119
4.4. Demostraciones 124
4.4.1. Demostración directa 124
4.4.2. Demostración por contradicción o reducción al absurdo 125
4.4.3. Demostración por inducción matemática 127

© Todos los derechos reservados

4.5. Verificación formal de programas a través de la lógica deductiva 131
4.5.1. Verificación formal de algoritmos mediante axiomas de la lógica de Hoare 131
4.5.2. Reglas de inferencia para verificar programas 132
4.5.3. Verificación formal de programas una mirada actual 137

5. Algoritmos

5.1. Introducción 141
5.2. Algoritmos 142
5.3. Máquina de Turing 152
5.4. Algoritmos recursivos 158
5.5. Complejidad de un algoritmo 161

Bibliografía

© Todos los derechos reservados

© TODOS LOS DERECHOS RESERVADOS

Presentación

La asignatura Lógica introduce los conceptos fundamentales sobre esta ciencia, los cuales sirven de base para las matemáticas, la inteligencia artificial, la programación y las ciencias de la computación en general, por solo citar algunos ejemplos de aplicación. El estudio de la Lógica permite el desarrollo del razonamiento lo cual le será útil tanto en el estudio del resto de las asignaturas de esta carrera como en su práctica profesional, especialmente en el diseño y la programación. A continuación se presenta un breve resumen de cada uno de los capítulos.

Capítulo	Objetivos particulares	Aportes y resultados conseguidos
1	• Definir los elementos básicos del lenguaje de la Lógica proposicional desde el punto de vista sintáctico y semántico. • Introducir las leyes de la Lógica proposicional y su uso para la demostración de equivalencias entre fórmulas.	• Se consigue definir los elementos sintácticos y semánticos que componen el lenguaje de la Lógica proposicional. • Se logra determinar el valor veritativo de las fórmulas a partir del uso de las tablas de verdad. • Se realizan demostraciones de equivalencias entre fórmulas.
2	• Diseñar y simplificar circuitos lógicos, teniendo en cuenta sus características y elementos fundamentales.	• Se consigue diseñar circuitos lógicos sencillos a partir del uso de compuertas lógicas. • Se consigue simplificar circuitos lógicos por el método algebraico y los mapas de Karnaugh.
3	• Definir los elementos básicos del lenguaje de la Lógica de predicados desde el punto de vista sintáctico y semántico.	• Se introduce el concepto de predicado y cuantificador, con algunos ejemplos para su entendimiento. • Se logra definir los elementos sintácticos y semánticos que componen el lenguaje de la Lógica de predicado. • Se determina el valor veritativo de fórmulas bajo una determinada interpretación.
4	• Aplicar el uso del razonamiento deductivo y la inducción en la realización de demostraciones.	• Se consigue caracterizar y aplicar el esquema de las estructuras deductivas, y su veracidad. • Se caracterizan las técnicas de demostración: directa, por contradicción y por inducción matemática. • Se logra aplicar las reglas de inferencia y la inducción matemática en la realización de demostraciones.

© Todos los derechos reservados

Capítulo	Objetivos particulares	Aportes y resultados conseguidos
5	• Diseñar algoritmos en pseudocódigo y determinar su complejidad temporal. • Describir el funcionamiento de una Máquina de Turing, sus características y formas de representación.	• Se describen las características y elementos fundamentales de los algoritmos. • Se consigue diseñar algoritmos sencillos haciendo uso de pseudocódigos. • Se consigue explicar el funcionamiento de los algoritmos recursivos a través de ejemplos concretos. • Se logra aplicar el modelo de Máquina de Turing en la solución de problemas sencillos. • Se logra explicar el procedimiento para determinar la complejidad de un algoritmo y se ilustra su aplicación en algunos de los algoritmos presentados en el capítulo.

© Todos los derechos reservados

Lógica proposicional 1

Objetivos

- Describir brevemente la evolución de la Lógica como ciencia.
- Analizar los conceptos básicos de la lógica proposicional, los operadores y las tablas de verdad; aplicándolos en situaciones concretas.
- Clasificar e interpretar fórmulas para la determinación de su valor veritativo.
- Definir las relaciones de equivalencia e implicación lógica, las leyes de la lógica proposicional y sus propiedades.
- Aplicar las leyes de la lógica proposicional para la demostración de equivalencias entre fórmulas.
- Valorar la importancia y uso de la lógica proposicional.

© Todos los derechos reservados

1.1. INTRODUCCIÓN

Etimológicamente la palabra lógica deriva de λογική (*logike*), término del griego antiguo, que significa «dotado de razón, intelectual, dialéctico, argumentativo», que a su vez proviene de λόγος (*logos*), «palabra, pensamiento, idea, argumento, razón o principio». La Lógica como ciencia, expone las leyes, modos y formas del conocimiento científico. (Real Academia Española, 2001).

Su objeto de estudio abarca la investigación, desarrollo y establecimiento de los principios fundamentales que rigen el análisis de la validez de un razonamiento o inferencia, independientemente del significado que este pueda tener para la conciencia en el contexto donde se haya emitido. La Lógica es la base del razonamiento matemático, se utiliza principalmente en la demostración de teoremas. En la electrónica se emplea en el diseño de circuitos mediante compuertas lógicas para la construcción de dispositivos electrónicos de cómputo, y en ciencias de la computación permite verificar que los programas funcionen correctamente.

Existen varios sistemas lógicos, entre ellos, los clásicos, los no clásicos y los modales. Este manual se centra en el estudio de la Lógica proposicional y la Lógica de predicados pertenecientes a la Lógica clásica. En el presente capítulo se realiza un análisis sintáctico y semántico del lenguaje de la Lógica proposicional, a partir de los elementos fundamentales que la componen.

La Lógica es la base de todo razonamiento matemático, y tiene aplicaciones prácticas en el diseño de equipos informáticos, la especificación de sistemas, la inteligencia artificial, la programación computacional, los lenguajes de programación y en otras áreas de ciencias de la computación, así como en otros muchos campos de estudio.

Para entender las matemáticas debemos entender qué es lo que constituye un argumento matemático correcto, es decir, una demostración. Además, para aprender matemáticas, una persona necesita construir activamente argumentos matemáticos, no limitarse a leer una exposición. En este capítulo se presentan herramientas para construir estos argumentos. Las demostraciones no son importantes sólo en matemáticas, sino en muchas partes de las ciencias de la computación, entre las que se incluyen verificación de programas, análisis de resultados de algoritmos y sistemas de seguridad. Se han construido sistemas de razonamiento automatizado que permiten a los ordenadores construir sus propias demostraciones.

© Todos los derechos reservados

1.2. BREVE HISTORIA DE LA EVOLUCIÓN DE LA LÓGICA COMO CIENCIA

Los inicios de la Lógica, como instrumento de análisis del razonamiento, datan de la Edad Antigua. China, India y Grecia fueron las tres civilizaciones donde se desarrolló originalmente, aunque esta última fue la de mayor trascendencia en la historia de la Lógica, con Aristóteles como su principal exponente.

El filósofo griego Aristóteles es considerado como el “padre de la Lógica”. Desarrolló métodos sistemáticos para analizar y evaluar argumentos que permitieran ratificar o rebatir un determinado pensamiento. Fue el primero en formalizar los razonamientos con el uso de letras en la representación de términos, sentó las bases de la Lógica proposicional y estableció procedimientos para determinar la verdad o falsedad de proposiciones compuestas. También investigó sobre los argumentos deductivos, base fundamental de la ciencia experimental. En sus escritos; conocidos como *Órganon*; aborda temas como la proposición, definición, prueba, falacia y trata ampliamente el silogismo.

En la Edad Media, filósofos y matemáticos como Descartes y Leibniz, con sus análisis en el campo de las matemáticas, marcan el posterior desarrollo de la Lógica. Leibniz trabajó en la realización de una máquina de cálculo a partir de un lenguaje universal especificado con precisión matemática, aunque sus intentos fracasaron la idea de un lenguaje matemático universal da paso al desarrollo de la Lógica en el siglo XX.

El italiano Giuseppe Peano (1858-1932) fue quien acuñó el término de "Lógica Matemática", plasmando en su obra sus principios y su aplicación práctica; junto a Boole, De Morgan y Frege; lleva a cabo una profunda transformación de la Lógica sobre la base del análisis matemático.

George Boole y Augustus De Morgan percibieron la similitud de las operaciones lógicas con las matemáticas y crean operadores lógicos equivalentes a los operadores aritméticos. Boole desarrolló un sistema algebraico capaz de trabajar la lógica como un cálculo en el que se pudieran aplicar operaciones matemáticas como la suma y la multiplicación, y en el que los valores de verdad se representan mediante 0 (falsedad) y 1 (verdad). Este sistema es conocido actualmente como "Álgebra de Boole" y tiene aplicaciones en el área de la teoría de la probabilidad y el diseño de circuitos digitales combinacionales. A Boole también se le atribuye la creación de las tablas de verdad para determinar la veracidad de una proposición compuesta. Por su parte, De Morgan publica en 1847 su obra Lógica formal, donde introduce las leyes de De Morgan e intenta generalizar la noción de silogismo.

A finales del siglo XIX y principios del siglo XX, Gottlob Frege elabora un sistema completo de Lógica de predicados al introducir los cuantificadores con su Teoría de la Cuantificación, dando así origen a la Lógica de primer orden. Sin embargo el sistema de Frege tenía algunas contradicciones que son resuelta por Bertrand Russell y Alfred North Whitehead en 1910, quienes codificaron la lógica simbólica en su presente forma, por esta razón se les atribuye a ellos la fundación de la lógica formal moderna.

A la formalización y desarrollo de la Lógica de predicado contribuyeron varios matemáticos y filósofos de esta etapa, entre ellos: Löwenheim, Hilbert, Ackermann, Gödel, Church, Turing, Tarski y Gentzen, este último con su propuesta de la deducción natural como una alternativa para la construcción axiomática de los sistemas lógicos.

© Todos los derechos reservados

1.3. INTRODUCCIÓN A LA LÓGICA PROPOSICIONAL

Entre los sistemas lógicos clásicos se encuentran: la Lógica proposicional, la Lógica de primer orden y la Lógica de segundo orden. En este epígrafe se abordan los principales elementos de la lógica proposicional, mientras que el capítulo 3 abarca la Lógica de primer orden. La lógica de segundo orden no es objetivo de este material.

La Lógica proposicional establece los métodos para la formación de proposiciones y las relaciones existentes entre ellas, además de evaluarlas para determinar sus valores de verdad y realizar inferencias de proposiciones (conclusiones) a partir de proposiciones iniciales supuestas como verdaderas (premisas). El estudio de la Lógica evita las ambigüedades del lenguaje pues no tiene en cuenta la estructura interna de las proposiciones, es decir, su significado en el lenguaje natural. Por ejemplo:

1. Fernando es novio de Rebeca o es novio de Ángela.
2. Fernando no es novio de Ángela.
3. Por lo tanto, Fernando es novio de Rebeca.

Los dos primeros enunciados constituyen las premisas, cuya veracidad permiten arribar a la conclusión expuesta en el tercer enunciado. Sin embargo la validez de este argumento no se debe al significado de las expresiones «Fernando es novio de Rebeca» y «Fernando es novio de Ángela», éstas expresiones podrían cambiarse por otras y el argumento permanecer válido, siempre que se cumpla que las premisas sean verdaderas y por tanto la conclusión también.

Para que se tenga una idea de la diferencia entre el lenguaje natural y la lógica, considere las oraciones siguientes:

a) Todos los seres vivos son mortales.

b) Ningún ser vivo es inmortal.

Estas dos oraciones son diferentes desde el punto de vista gramatical, sin embargo, se puede apreciar intuitivamente que ambas expresan la misma idea, aun utilizando oraciones con sus traducciones en otros idiomas. La Lógica establece la equivalencia entre a) y b) como se verá más adelante.

Las proposiciones

En todo proceso que involucre la conciencia y el razonamiento humano existe un flujo de información, ya sea como parte de un acto de comunicación, o de asimilación de conocimiento mientras se lee, observa o escucha.

Desde cierto punto de vista podemos decir que vivimos inmersos en un mundo de información, de la cual no solo percibimos la que nos interesa, pero sí solo procesamos aquella que de cierta forma “entendemos”.

La información que nos resulta inteligible posee una determinada estructura basada en un lenguaje que podemos interpretar. Este lenguaje puede ser el lenguaje natural con el que usualmente nos comunicamos los humanos, podría ser también un lenguaje científico como el lenguaje matemático o un lenguaje de programación, por citar algunos ejemplos.

Durante el proceso de interpretación de una expresión de un determinado lenguaje, se llevan a cabo en el razonamiento humano un procesamiento de la información en tres niveles fundamentales: el nivel sintáctico, el semántico y el pragmático. El nivel sintáctico es el nivel básico de procesamiento de información y es donde se determina si la expresión lingüística está formada de forma correcta o no. Esto equivaldría en el lenguaje natural a determinar si en una oración todas las palabras pertenecen al idioma, y si los verbos, los sustantivos, las preposiciones y demás elementos del lenguaje están bien usados de acuerdo con las reglas gramaticales del idioma.

Una vez procesada la expresión en el nivel sintáctico el razonamiento humano trata de darle un significado y es cuando entra en juego entonces el nivel de interpretación semántico.

El nivel pragmático está ligado a determinar la intención con que ha sido formulada una determinada expresión, en lo cual intervienen factores como la conciencia humana.

© Todos los derechos reservados

La Gramática se encarga del estudio de los dos primeros niveles de interpretación. De modo que es relevante para la gramática la estructura de las expresiones lingüísticas (Teoría sintáctica) y su significado (Teoría semántica). Para la Lógica es relevante este segundo punto, pero no en toda su extensión, sino solo la dimensión que permite determinar al razonamiento humano si un determinado enunciado es correcto o no, o lo que a los efectos lógicos es equivalente, si un determinado enunciado es verdadero o falso, independientemente del significado que este pueda tener para la conciencia en el contexto donde se haya emitido.

No podría determinarse la veracidad o falsedad de todo tipo de enunciado que pueda ser formulado en un determinado lenguaje, pues para algunos el carácter de ser verdadero o falso simplemente podría no tener sentido. De modo que solo nos interesaría un tipo especial de enunciado que llamaremos proposiciones y así comenzaremos con el estudio de la Lógica proposicional.

Los matemáticos usan la lógica, para demostrar teoremas e inferir resultados que puedan ser aplicados en investigaciones. En la computación, para revisar programas y crear sus algoritmos, es utilizada en el diseño de computadoras. Existen circuitos integrados que realizan operaciones lógicas con los bits, gracias a estos se ha desarrollado las telecomunicaciones (telefonía móvil, internet, etc.)

© Todos los derechos reservados

¿Qué es una proposición?

Una de las estructuras más importantes de nuestro razonamiento son las proposiciones. A su vez es necesario un sistema lógico para determinar la validez de nuestros razonamientos, la Teoría Semántica aporta este sistema.

Las proposiciones son oraciones que tienen una connotación lógica: las proposiciones expresan información, conocimiento resultado de nuestra actividad pensante.

Una proposición es una oración declarativa del lenguaje natural, llamadas también oraciones enunciativas o aseverativas, que se utilizan para comunicar o informar algo que ocurre, ha ocurrido u ocurrirá. Una propiedad fundamental de las proposiciones declarativas es que tienen un *valor de verdad* o *valor veritativo*. Este material se centra en la Lógica proposicional bivalente donde las proposiciones solo pueden ser verdaderas o falsas, pero no las dos a la vez y no dependen de una opinión individual.

**Nota:* las frases interrogativas, dubitativas, imperativas o exclamativas no son consideradas como proposiciones.

Ejemplo 1.3.1.

Las siguientes oraciones declarativas son proposiciones:

a) La lógica es un lenguaje formal.

b) El fútbol es el más universal de los deportes.

c) ***Si*** apruebas el examen, ***entonces*** hacemos la fiesta.

d) La capital de Ecuador es Quito.

e) El 7 es un número impar y es primo.

f) Bruselas es la capital de la Unión Europea.

g) 1 + 1 = 3

h) 3 es un número primo, pero 4 no.

EJEMPLO 1.3.2.

Las siguientes oraciones declarativas no son proposiciones:

a) ¿Te gusta la informática?

b) Apriete F5 para actualizar su navegador.

c) Lee esto con atención.

d) $y + 1 = 3$

e) $x + y = z$

Las frases a) y b) no son proposiciones porque no son declarativas. Las frases d) y e) no son proposiciones porque no son ni verdaderas ni falsas, ya que no se les han asignado valores a las variables.

© TODOS LOS DERECHOS RESERVADOS

Para denotar proposiciones usamos letras, al igual que usamos letras para denotar variables. Por convenio, las letras que se utilizan para denotar proposiciones son *p, q, r, s,...* El *valor de verdad* de una proposición es verdadero, y se denota por 1, si es una proposición verdadera, o falso, denotado por 0, si es una proposición falsa.

El área de la lógica que trata de proposiciones se llama ***cálculo proposicional*** o ***lógica proposicional***. Fue desarrollada sistemáticamente por primera vez por el filósofo griego Aristóteles hace más de dos mil trescientos años.

Definición 1.3.1: una proposición p es una ***proposición simple*** si la misma no puede descomponerse en otras proposiciones.

Definición 1.3.2: si la proposición q surge de la negación de una proposición simple p_1 o si las proposiciones simples p_1, p_2,... p_n se combinan para formar la proposición q, diremos que q es una ***proposición compuesta***.

EJEMPLO 1.3.3.

Las siguientes proposiciones son simples:

a) El conjunto $A = \{b, c, d\}$ es igual al conjunto $B = \{d, c, b\}$.

b) España es un país transcontinental.

c) España es un Estado miembro de la Unión Europea.

d) El conjunto $B = \{d, c, b\}$ es subconjunto de $A = \{b, c, d\}$.

EJEMPLO 1.3.4.

Las siguientes proposiciones son compuestas:

a) La matriz $\begin{pmatrix} 1 & 3 \\ 2 & 6 \end{pmatrix}$ ***no*** es inversible.

b) España es un país transcontinental y es un Estado miembro de la Unión Europea.

c) Si el conjunto $A = \{b, c, d\}$ es igual al conjunto $B = \{d, c, b\}$ **entonces** $A \subseteq B$.

Definición 1.3.3: una variable proposicional es una proposición arbitraria con un valor de verdad no especificado, es decir, puede ser verdadera o falsa.

EJEMPLO 1.3.5.

A continuación se muestran ejemplos de variables proposicionales *q, r,* y s:

q: Si la computadora enciende entonces comienzo a programar.

r: Realizo el algoritmo en clases o voy hacia la biblioteca a estudiar.

s: Carlos es disciplinado o no entiende al profesor.

En la lógica proposicional, sustituiremos el contenido de las proposiciones por variables proposicionales. Toda variable proposicional q, puede ser sustituida por cualquier enunciado siendo sus posibles estados, verdadero o falso.

Cuando q es verdadera se dice que q es lógicamente equivalente a 1 y es escribe $q \cong 1$.

Cuando q es falsa se dice que q es lógicamente equivalente a 0 y se escribe $q \cong 0$.

© TODOS LOS DERECHOS RESERVADOS

1.4. SINTAXIS DE LA LÓGICA PROPOSICIONAL

Establece los elementos que componen el lenguaje (alfabeto o vocabulario), así como las normas de escritura que permiten relacionarlos correctamente entre sí en correspondencia con el lenguaje de la lógica (gramática).

1.4.1. ALFABETO O VOCABULARIO

El alfabeto de un sistema formal es el conjunto de símbolos que conforman el lenguaje del sistema, para la Lógica proposicional estos son:

- Variables proposicionales:

  Son un conjunto de variables que se utilizan para representar proposiciones simples. En general se toman del alfabeto latino, empezando por la letra p hasta llegar a la letra *z*, en caso de ser necesario se pueden utilizar subíndices p_1, p_2,... p_n. Para representar las proposiciones se utiliza la notación siguiente:

  p: La Lógica es un lenguaje formal

  De esta forma se indica que la variable p representa la proposición "La Lógica es un lenguaje formal".

- Constantes proposicionales:

  Se emplean para denotar el valor de verdad de una proposición, de esta forma se utilizará **1** para especificar las proposiciones que son verdaderas y **0** para las que son falsas.

- Operadores o conectores lógicos:

  Prestamos ahora atención a los métodos para producir proposiciones nuevas a partir de las ya existentes. Estos métodos fueron estudiados por el matemático inglés George Boole en 1854 en su libro "Las leyes del pensamiento". Muchos enunciados matemáticos se construyen combinando una o más proposiciones. Las nuevas proposiciones, llamadas fórmulas o proposiciones compuestas, se forman a partir de las existentes usando operadores lógicos.

  Como se mencionó anteriormente, las proposiciones complejas utilizan determinados elementos para enlazar proposiciones simples, estos elementos se denominan operadores o conectores lógicos, similares a los conectores del lenguaje natural. La tabla 1.1 muestra los diferentes operadores lógicos con su símbolo y significado, así como un ejemplo de su uso.

© TODOS LOS DERECHOS RESERVADOS

Tabla 1.1: Operadores lógicos.

Operador	Símbolo	Lenguaje natural	Ejemplo
Negación	$\neg$	Representa las expresiones del lenguaje natural «no», «no es cierto», «es falso que», «no es posible» o cualquier otra del lenguaje natural que indique negación.	*p:* Me gusta la música. $\neg p$: No me gusta la música.
Conjunción	$\wedge$	Representa las expresiones del lenguaje natural «y», «ni», «pero», «mas» o cualquier otra del lenguaje natural utilizada como conjunción copulativa. Significa que se cumplen todas las proposiciones unidas por la conjunción.	*p:* Me gusta la música. *q:* Me gusta la literatura. $p \wedge q$: Me gusta la música y la literatura.
Disyunción	$\vee$	Representa las expresiones del lenguaje natural «o», «u» y cualquier otra del lenguaje natural utilizada como disyunción. Significa que se cumple al menos una de las proposiciones unidas por la disyunción. Este operador tiene un sentido inclusivo, es decir, acepta que se cumplan todas las proposiciones.	*p:* Me gusta la música. *q:* Me gusta la literatura. $p \vee q$: Me gusta la música o la literatura.
Condicional	$\rightarrow$	Representa las expresiones del lenguaje natural «si», «si...entonces», «cuando», «porque», «con tal que», «...por tanto...» o cualquier otra del lenguaje natural que exprese una condición necesaria o suficiente. Significa que una de las proposiciones ocurre como consecuencia de la ocurrencia de la otra. Nota: la proposición delante del operador se denomina *antecedente* y la que le sigue, *conclusión*.	*p:* Me gusta la música. *q:* Me gusta la literatura. $p \rightarrow q$: Me gusta la música, por tanto me gusta la literatura.
Bicondicional	$\leftrightarrow$	Representa las expresiones del lenguaje natural «... si y solo si...», «... equivale a...», «... es igual a...», «... es lo mismo que...» o cualquier otra del lenguaje natural que exprese equivalencia, condición necesaria y suficiente. Significa que se cumplen ambas proposiciones o ninguna de las dos.	*p:* Me gusta la música. *q:* Me gusta la literatura. $p \leftrightarrow q$: Me gusta la música si y solo si me gusta la literatura.

- Signos auxiliares de escritura:

  Como signos auxiliares se emplean los paréntesis, que se utilizan para especificar a qué proposiciones se aplica un determinado operador, evitando así ambigüedades en la lectura.

© Todos los derechos reservados

EJEMPLO 1.4.1. TRABAJO CON OPERADORES

Formalice las proposiciones siguientes:

a) El ejercicio del examen estaba largo y complicado.

b) Será mejor que nunca te emborraches.

c) Me divierto mucho cuando Frank hace chistes.

d) No me gusta la Física ni la Matemática.

e) El triángulo es equilátero si, y solo si, sus tres lados son iguales.

f) Juan y María se casaron.

g) Solo si Pedro está saludable, vamos a la playa.

h) Jugamos bajo la lluvia, o nos quedamos en casa y vemos una película.

Solución:

a) **Nota*: en el lenguaje natural la conjunción "*y*" evita repetir expresiones para otorgar diferentes cualidades a un mismo objeto, por lo que en este enunciado "*complicado*" no constituye una proposición, en este caso la proposición sería "*El ejercicio del examen estaba complicado*" que especifica el objeto al que se atribuye esta cualidad.

p_1: El ejercicio del examen estaba largo.

q1: El ejercicio del examen estaba complicado.

$p_1 \wedge q_1$

b) **Nota*: en este enunciado, "nunca", cumple la función de la cláusula "no" por lo que se utiliza el operador negación.

$\neg p_2$: Será mejor que te emborraches.

c) p_3: Me divierto mucho.

q_3: Frank hace chiste.

$q_3 \rightarrow p_3$ *Nota: en este enunciado, "*cuando*", cumple la función de la cláusula "*si*" por lo que q_3 representa el antecedente o hipótesis y p_3 la conclusión.

d) **Nota*: "*ni*" expresa negación en aquellos enunciados que contienen una conjunción "*y*" por lo que se utiliza tanto el operador negación como la conjunción.

p_4: Me gusta la Física.

q_4: Me gusta la Matemática.

$\neg p_4 \wedge \neg q_4$

e) p_5: El triángulo es equilátero.

q_5: El triángulo tiene sus tres lados iguales.

$q_5 \leftrightarrow p_5$

f) **Nota*: estaría incompleto decir "Juan se casó" o "María se casó" puesto que el enunciado expresa que el casamiento fue entre ambos y no por separado, en este

© TODOS LOS DERECHOS RESERVADOS

caso "y", no representa una conjunción entre dos proposiciones simples sino una relación que se formaliza como una proposición simple.

p_6: Juan y María se casaron.

g) **Nota*: existen diferencias entre condiciones necesarias y condiciones suficientes. Una condición suficiente expresa que lo; condicionado puede ser verdadero aunque no sea verdadera la condición, por tanto, la condición es el antecedente y lo condicionado la conclusión. Con las condiciones necesarias no puede ser verdadero lo condicionado si no es verdadera la condición, por lo que en estos casos la condición pasa a ser la conclusión. Algunas expresiones para diferenciarlas son:

Condición suficiente: si... entonces...; basta... para que...; es suficiente que...

Condición necesaria: sólo si... entonces...; ...sólo en el caso en que...; es necesario... para que...; sin esto... no se podría...

El enunciado del inciso *g)* expresa una condición necesaria donde "*solo si*" es la cláusula que antecede a la conclusión, de esta forma "*Solo si Pedro está saludable, vamos a la playa*" se considera desde el punto de vista de la lógica lo mismo que "*Si vamos a la playa entonces Pedro está saludable*", por lo que *p7* representa el antecedente y *q7* la conclusión.

p_7: Vamos a la playa.

q_7: Pedro está saludable.

$p_7 \rightarrow q_7$

Si el enunciado hubiese sido "*Si Pedro está saludable, vamos a la playa*", q_7 representaría el antecedente y p_7 la conclusión.

p_7: Vamos a la playa.

q_7: Pedro está saludable.

$q_7 \rightarrow p_7$

h) p_8: Jugamos bajo la lluvia.

q_8: Nos quedamos en casa.

r_8: Vemos una película.

$p_8 \vee (q_8 \wedge r_8)$

1.4.2. Gramática. Fórmulas bien formadas

La gramática define la sintaxis del lenguaje, estableciendo todas las posibles combinaciones que se pueden formar con los símbolos del alfabeto, las cuales se denominan fórmulas y están sujetas a las reglas siguientes:

1. Toda constante o variable proposicional es una fórmula.
2. Si A es una fórmula, entonces ¬A es también una fórmula.
3. Si A y B son fórmulas, entonces $(A \vee B)$, $(A \wedge B)$, $(A \rightarrow B)$ y $(A \leftrightarrow B)$ son fórmulas.

© Todos los derechos reservados

4. No hay fórmulas en el lenguaje proposicional que no sean las resultantes de combinaciones de las reglas 1, 2 y 3.

Los símbolos A y B usados para definir las reglas de formación de fórmulas no pertenecen al alfabeto proposicional, son símbolos del lenguaje natural utilizados para hacer referencia a cualquier fórmula del lenguaje proposicional.

EJEMPLO 1.4.2. FÓRMULAS BIEN FORMADAS

Determine si las siguientes constituyen fórmulas bien formadas (FBF):

a) $((p \rightarrow q) \vee r))$

b) $\neg r$

c) $(((p \rightarrow q) \vee r) \wedge s)$

d) $((p \leftrightarrow q) \wedge r)$

e) $(q \wedge (p \vee))$

f) $((p \leftrightarrow q)\ (r \wedge s))$

g) $p\ (\rightarrow q \vee r))$

h) $\neg(r \wedge s)$

i) $((p \vee q) \leftrightarrow (p \wedge s))$

Solución:

La regla 1 asegura que ***p, q, r*** y ***s*** constituyen FBF en cada uno de los incisos, luego:

a) Aplicando la regla 3, $(p \rightarrow q)$ es una FBF que para mayor claridad en el análisis se reemplazará por el símbolo A_0, por lo que la fórmula general, a la que se denominará A, queda de la siguiente forma $A = (A_0 \vee r))$. Como se puede apreciar, A no es una FBF ya que tiene un paréntesis de más, la formulación correcta sería $((p \rightarrow q) \vee r)$.

b) Con la regla 2 se puede comprobar que $\neg r$ no es una FBF puesto que sobran los paréntesis, la formulación adecuada sería: $\neg r$.

c) Como se vio en la corrección del inciso *a*, $((p \rightarrow q) \vee r)$ es una FBF, sustituyéndola por el símbolo A_0 quedaría como $(A_0 \wedge s)$, la cual también es una FBF según la regla 3.

d) A partir de la regla 2, $\neg q$ es una FBF, que combinada con la regla 3 se puede decir que $(p \leftrightarrow \neg q)$ es una FBF, remplazándola por A_0 quedaría $(A_0 \wedge r)$ que es una FBF según la propia regla 3.

e) $(p \vee)$ no es una FBF pues no cumple con ninguna de las reglas establecidas, el operador $\vee$ debe estar aplicado entre dos proposiciones, por tanto la fórmula $(q \wedge (p \vee))$ no es una FBF.

f) Siguiendo la regla 3, $A_0 = (p \leftrightarrow q)$ y $A_1 = (r \wedge s)$ son FBF, sin embargo $(A_0\ A_1)$ no lo es puesto que falta declarar un operador entre las fórmulas A_0 y A_1.

© TODOS LOS DERECHOS RESERVADOS

g) Según la regla 3, $(q \vee r)$ es una FBF, si $A_0 = (q \vee r)$ entonces p $(\rightarrow A_0)$ que no constituye una FBF pues no cumple con las reglas establecidas. Una posible corrección sería $(p \rightarrow (q \vee r))$.

h) $(r \wedge s)$ es una FBF que cumple con la regla 3, siendo $A_0 = (r \wedge s)$ luego $\neg A_0$ es también una FBF en correspondencia con la regla 2. Por tanto $\neg(r \wedge s)$ es una FBF.

i) Aplicando la regla 3, $A_0 = (p \vee q)$ y $A_1 = (p \wedge s)$ son FBF, utilizando esta misma regla y sustituyendo, $(A_0 \leftrightarrow A_1)$ es también una FBF, entonces $((p \vee q) \leftrightarrow (p \wedge s))$ es una FBF.

Podemos construir fórmulas usando el operador negación y los operadores lógicos definidos hasta el momento. Generalmente, utilizaremos paréntesis para especificar el orden en el que deben aplicarse los operadores lógicos en una fórmula. Por ejemplo, $(p \vee q) \wedge (\neg r)$ es la conjunción de $p \vee q$ y $\neg r$. Sin embargo, para reducir el número de paréntesis, especificamos que el operador negación se aplica antes que los operadores lógicos. Esto significa que el operador negación $\neg p \wedge q$ es la conjunción de $\neg p$ y q, es decir, $(\neg p) \wedge q$ no la negación de la conjunción de p y q, es decir $\neg(p \wedge q)$.

Otra regla general de precedencia es que el operador conjunción precede siempre al operador disyunción, de tal forma que $p \wedge q \vee r$ significa $(p \wedge q) \vee r$ y no $p \wedge (q \vee r)$. Debido a que esta regla es difícil de recordar, en el texto continuaremos usando paréntesis para que quede claro el orden utilizado en los operadores conjunción y disyunción.

Finalmente, es una regla aceptada que el operador condicional $\rightarrow$ y bicondicional $\leftrightarrow$ tienen precedencia inferior que los operadores conjunción y disyunción, $\wedge$ y $\vee$. Consecuentemente, $p \vee q \rightarrow r$ es lo mismo que $(p \vee q) \rightarrow r$. Usaremos paréntesis cuando el orden de los operadores condicional y bicondicional se deba tener en cuenta, aunque el operador condicional tiene precedencia sobre el bicondicional. La tabla 1.2 muestra los niveles de precedencia de los operadores lógicos.

Tabla 1.2: Jerarquía en orden descendente de prioridades para los operadores lógicos.

Operador		Precedencia
Negación	$\neg$	1
Conjunción	$\wedge$	2
Disyunción	$\vee$	3
Condicional	$\rightarrow$	4
Bicondicional	$\leftrightarrow$	5

© Todos los derechos reservados

1.5. SEMÁNTICA DE LA LÓGICA PROPOSICIONAL

La semántica se encarga de estudiar la relación de los signos con su significado, es decir, atribuye significados (Verdadero o Falso) a las distintas fórmulas del lenguaje, utilizando para esto las tablas de verdad.

1.5.1. TABLAS DE VERDAD

El conjunto de los posibles valores de una proposición, los representaremos en las llamadas tablas de verdad, ideadas por Ludwing Wittgenstein en 1921.

Las tablas de verdad permiten determinar los valores de verdad de las proposiciones compuestas a partir de los conectores (operadores lógicos) utilizados y de los valores de verdad de sus proposiciones simples. Para otorgar los valores de verdad a una proposición se utilizan las constantes proposicionales introducidas en el epígrafe 1.4.1, siendo 1 el valor verdadero y 0 el valor falso.

Definición 1.5.1: la tabla de verdad de una proposición compuesta enumera todas las posibles interpretaciones de los valores de verdad para las proposiciones $p_1, p_2,..., p_n$.

EJEMPLO 1.5.1.

Si q es una proposición compuesta por 2 proposiciones simples (p_1, y p_2), entonces la tabla de verdad de q recoge los siguientes valores de verdad dentro de sus 4 interpretaciones:

Tabla 1.3: Tabla de verdad para dos variables proposicionales.

p1	***p2***
1	1
1	0
0	1
0	0

Nótese como cada interpretación no es más que las posibles combinaciones de valores de verdad de las proposiciones simples $p_1, p_2,..., p_n$.

© TODOS LOS DERECHOS RESERVADOS

EJEMPLO 1.5.2.

Si q es una proposición compuesta por 3 proposiciones simples (p_1, p_2 y p_3), entonces la tabla de verdad de q recoge los siguientes valores de verdad dentro de sus 8 interpretaciones:

Tabla 1.4: Tabla de verdad para tres variables proposicionales.

p1	*p2*	*p3*
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Nótese como las tablas de verdad guardan mucha relación con las tablas de pertenencia de la Teoría de conjuntos.

© TODOS LOS DERECHOS RESERVADOS

Existe una importante propiedad de las tablas de verdad la misma expresa que: Si q es una proposición compuesta por n proposiciones simples ($n \geq 1$), entonces la tabla de verdad de q tiene un total de 2^n interpretaciones.

EJEMPLO 1.5.3.

Si q es una proposición compuesta por 4 proposiciones simples (p_1, p_2, p_3 y p_4), entonces la tabla de verdad de q posee $2^4 = 16$ interpretaciones.

Existen disímiles maneras de conectar proposiciones entre sí. A continuación se muestran los operadores ***proposicionales*** básicos que permiten formar proposiciones compuestas.

A continuación se muestran las tablas de verdad de los diferentes operadores lógicos.

- **Negación:** toma valor 1 si la proposición tiene valor 0, y es 0 cuando la proposición tiene valor 1.

Definición 1.5.2: sea p una proposición. El enunciado "no p" es otra proposición, llamada la **negación** de p. La negación p de se denota mediante $\neg p$. Si $p \cong 1$ entonces $\neg p \cong 0$ y si $p \cong 0$ entonces $\neg p \cong 1$ y $\neg p \cong 1$.

Tabla 1.5: Tabla de verdad del operador Negación.

p	$\neg p$
1	0
0	1

En el lenguaje natural que conocemos, algunas de las frases más usadas para denotar la negación de cualquier proposición p son:

- No ***p***.
- Es falso que ***p***.
- No es cierto que ***p***.
- No se cumple que ***p***.

EJEMPLO 1.5.4.

Sean las siguientes proposiciones:

q: La relación $S = \{(1, 2), (2, 2), (2, 3)\}$ es una relación simétrica.

$\neg q$: La relación $S = \{(1, 2), (2, 2), (2, 3)\}$ no es una relación simétrica.

Conocemos por lo estudiado en relaciones binarias que q es falsa (0) y que $\neg q$ es verdadera (1). Otra manera de escribir $\neg q$ es:

$\neg q$: No es cierto que la relación $S = \{(1, 2), (2, 2), (2, 3)\}$ es una relación simétrica.

- **Conjunción:** toma valor 1 solo si ambas proposición tienen valor 1, en caso contrario es 0.

  Definición 1.5.3: sean p y q proposiciones. La proposición "***p*** y ***q***", denominada conjunción de p y q, y denotada como $p \wedge q$, es la proposición que es verdadera cuando p y q son verdaderas y falsa en cualquier otros caso.

Tabla 1.6: Tabla de verdad del operador Conjunción.

p	q	$p \wedge q$
1	1	1
1	0	0

© TODOS LOS DERECHOS RESERVADOS

0	1	0
0	0	0

En el lenguaje natural que conocemos, algunas de las frases más usadas para denotar la conjunción entre dos proposiciones son:

- *p* y *q*.
- *p* pero *q*.
- *p* sin embargo *q*.
- *p* no obstante *q*.
- *p* a pesar de *q*.
- *p* además de *q*.
- *p* en adición a *q*.

EJEMPLO 1.5.5.

Sean las siguientes proposiciones:

$$p: \lim_{\chi \to 3} \frac{\chi^2 - 1}{\chi + 1} = 2$$

$$q: 3^2 - 1 = 8$$

Por tal motivo:

$$p \wedge q: \lim_{x \to 3} \frac{\chi^2 - 1}{\chi + 1} = 2 \text{ y } 3^2 - 1 = 8$$

Otra manera de expresarlo es:

$$p \wedge q: \lim_{x \to 3} \frac{\chi^2 - 1}{\chi + 1} = 2 \text{ a pesar de que } 3^2 - 1 = 8$$

- **Disyunción:** toma valor 1 si al menos una de las proposiciones tiene valor 1 y 0 cuando ambas proposición tiene valor 0.

  Definición 1.5.4: sean *p* y *q* proposiciones. La proposición "***p*** o ***q***", denominada disyunción de *p* y *q*, y denotada como $p \vee q$, es la proposición que es verdadera cuando *p* o *q* son verdaderas y únicamente falsa cuando ambas son falsas.

© TODOS LOS DERECHOS RESERVADOS

Tabla 1.7: Tabla de verdad del operador Disyunción.

p	q	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0

El uso del conectivo lógico o en una disyunción se asocia al significado en sentido inclusivo de la palabra *o**. Una disyunción es verdadera cuando al menos una de las dos proposiciones es verdadera. Por ejemplo, el o en sentido inclusivo se emplea en el enunciado:

«*Los estudiantes que hayan cursado Matemática I o Programación I pueden matricularse en esta clase.*»

Con esta frase se quiere decir que los estudiantes que han cursado bien Matemática I o bien Programación I pueden matricularse en la clase, así como los estudiantes que han cursado ambas asignaturas. Por otra parte, estamos usando el o exclusivo cuando decimos:

«*Los estudiantes que hayan cursado Matemática I o Programación I, pero no ambos, pueden matricularse en esta clase.*»

Ahora se quiere expresar que aquellos que hayan cursado tanto Matemática I como Programación I no pueden matricularse. Sólo pueden hacerlo aquellos que hayan cursado exactamente una de las dos asignaturas.

De forma similar, cuando en un menú de un restaurante vemos «*Se sirve sopa o ensalada como entrante*», casi siempre se quiere decir que los clientes pueden tomar bien sopa o bien ensalada, pero no ambos. Por tanto, éste es un uso exclusivo no inclusivo de la disyunción o.

En el lenguaje natural que conocemos, algunas de las frases usadas para denotar la disyunción entre dos proposiciones son:

- *p* o *q*.
- o *p* o *q* o ambas cosas.
- al menos *p* o *q*.
- como mínimo *p* o *q*.

Ejemplo 1.5.6.

Sean las siguientes proposiciones:

p: Carlos Miguel es un estudiante de la Universidad.

© Todos los derechos reservados

q: Carlos Miguel no es de primer año.

Por tal motivo:

$p \vee q$: Carlos Miguel es un estudiante de la Universidad o no es de primer año.

Otra manera de expresarlo es:

$p \vee q$: Carlos Miguel es un estudiante de la Universidad o no es de primer año o ambas cosas

- **Disyunción exclusiva:** esta proposición es verdadera cuando p es verdadera y q falsa y cuando es falsa y verdadera. Es falsa cuando tanto como son falsas y cuando ambas son verdaderas.

  Definición 1.5.5: sean p y q proposiciones. La proposición "***p o q pero no ambas cosas***", se denominada ***disyunción exclusiva*** de p y q, y denotada como $p \oplus q$, es la proposición que es verdadera cuando p es verdadera o q es verdadera pero no ambas a la vez.

  A continuación se muestra la tabla de verdad de la proposición compuesta $p \oplus q$.

p	q	$p \oplus q$
1	1	0
1	0	1
0	1	1
0	0	0

  En el lenguaje natural que conocemos, algunas de las frases usadas para denotar la disyunción exclusiva entre dos proposiciones son:

  – p o q pero no ambas cosas.
  – solamente p o solamente q.
  – como máximo p o como máximo q.

EJEMPLO 1.5.7.

Sean las siguientes proposiciones:

p: Remberto programa en C#.

q: Remberto programa en Java.

Por tal motivo:

$p \oplus q$: Remberto programa en C# o programa en Java pero no en ambas cosas.

© TODOS LOS DERECHOS RESERVADOS

Otra manera de expresarlo es:

$p \oplus q$: Remberto programa solamente en C# o solamente en Java

- **Condicional:** toma valor 0 si la causa (p, en la tabla 1.8) tiene valor 1 y la consecuencia (q, en la tabla 1.8) tiene valor 0, en el resto de los casos su valor es 1.

  En muchas definiciones o teoremas que hemos estudiado en este y otros cursos es necesario que se cumpla alguna condición para arribar a alguna conclusión.

  Definición 1.5.6: sean p y q proposiciones. La proposición "***p entonces q***", denominada ***condicional o implicación*** de p y q, y denotada como $p \rightarrow q$, es la proposición que es falsa cuando p es verdadera y q es falsa, en todos los demás casos es verdadera. En esta implicación, p se llama hipótesis (o premisa o condición) y q se llama tesis (o conclusión).

Tabla 1.8: Tabla de verdad del operador Condicional.

p	q	$p \rightarrow q$
1	1	1
1	0	0
0	1	1
0	0	1

Debido a que las implicaciones desempeñan un papel esencial en el razonamiento matemático, existen muchas formas de expresar $p \rightarrow q$. Encontrarás muchas de ellas, si no todas, entre las siguientes expresiones:

- Si p, entonces q.
- p implica q.
- Si p, q.
- p sólo si q.
- p es suficiente para q.
- Una condición suficiente para q es p.
- q si p.
- q siempre que p.
- q cuando p.
- q es necesario para p.
- Una condición necesaria para p es q.
- q se deduce de p.

© Todos los derechos reservados

La implicación $p \rightarrow q$ es falsa sólo en el caso de que p sea verdadera y q sea falsa. Es verdadera cuando tanto p como q son verdaderas y cuando p es falsa (no importa el valor de verdad de q).

Una forma útil de entender el valor de verdad de una implicación es pensar en una obligación o en un contrato. Por ejemplo, considera una afirmación en la que un profesor dice:

Si consigues el ciento por ciento de la puntuación en el examen final, sacarás un sobresaliente.

Si consigues completar correctamente el ciento por ciento de las preguntas, entonces podrías esperar sacar la máxima calificación. Si no consigues el ciento por ciento, puedes o no sacar un sobresaliente dependiendo de otros factores. En cualquier caso, si completas el ciento por ciento, pero el profesor no te pone un sobresaliente, te sentirás engañado.

Muchas personas encuentran confuso el hecho de que p sólo si q exprese lo mismo que si p entonces q. Para recordar esto, ten en cuenta que «p sólo si q» dice que p no puede ser verdadera cuando q no es verdadera. Esto es, el enunciado es falso si p es verdadera, pero q es falsa. Cuando p es falsa, q puede ser bien verdadera o bien falsa, porque la afirmación no dice nada acerca del valor de verdad de q.

Hay otras muchas maneras de expresar una condicional, lo único que debe tenerse claro para reconocerla es que la frase debe afirmar que en todos los casos que se cumpla p debe cumplirse también q (o sea que el cumplimiento de la condición p es suficiente para afirmar que la conclusión q tiene que cumplirse también). Dicho de otra manera, si q no se cumplió no es posible que p se haya cumplido (o sea que el cumplimiento de la conclusión q es necesaria para afirmar que la condición p tiene que cumplirse también.)

Resumiendo, la ***condición p es lo suficiente*** mientras que la ***conclusión q es lo necesario***.

La construcción *si-entonces* se usa en muchos lenguajes de programación de forma diferente que en lógica. La mayoría de los lenguajes de programación contienen sentencias como ***if*** *p* ***then*** *S*, donde p es una proposición y S un segmento de programa (una o más sentencias sintácticamente bien construidas que deben ser ejecutadas). Cuando la ejecución del programa encuentra tal sentencia, se ejecuta S si p es verdadera, pero S no se ejecuta si p es falsa.

EJEMPLO 1.5.8.

Hay frases que hacen más énfasis en la condición y otras en la conclusión. Entre las frases en español que denotan una proposición condicional están:

- "Si estudio, apruebo Lógica" en esta la coma hace función de "entonces", se identifica la condición p= "estudio" y la conclusión q= "apruebo Lógica". El esquema de esta frase es: "si *p, q*".

© TODOS LOS DERECHOS RESERVADOS

- "Apruebo Lógica si estudio", este ejemplo vuelve a ser similar y solo se han movido de lugar las proposiciones pero sigue siendo suficiente la condición *p*= "estudio" para afirmar la conclusión *q*= "apruebo Lógica". El modelo de esta frase es "*q* si *p*".
- "Solo si apruebo Lógica, estudio", el análisis de esta frase es similar a la anterior pero solo se ha cambiado de orden las proposiciones y por esto se ha agregado la coma. Se está afirmando de manera explícita que es necesaria la proposición *q*= "apruebo Lógica" para afirmar la condición *p*= "estudio". El esquema de esta frase es "solo si *q*, *p*".
- "Cuando estudio ocurre que apruebo Lógica", aquí el "cuando" está fijando una condición que es suficiente (la condición *p*= "estudio" para que se cumpla la proposición *q*= "apruebo Lógica". El esquema de esta frase es "cuando *p* ocurre *q*".
- "Apruebo Lógica cuando estudio", esta es similar a la anterior sólo variando el orden pero se está fijando una condición que es suficiente (la condición *p*= "estudio") para que se cumpla la proposición *q*= "apruebo Lógica". El esquema de esta frase es "*q* cuando *p*".
- "Basta que estudie para que apruebe Lógica", aquí se deja claro que es suficiente (basta) –"que estudie" y el esquema es "basta *p* para *q*".
- "No estudio a menos que apruebe Lógica", aquí se vuelve a dejar claro la necesidad de *q* para que ocurra *p* y el esquema es "no *p* a menos que *q*".

- **Recíproco, contrarrecíproco e inversa**

El ***recíproco*** de una proposición condicional es el resultado de cambiar los papeles de las variables que intervienen en la proposición, es decir, para $p \rightarrow q$ el recíproco es $\boldsymbol{q \rightarrow p}$.

Por otra parte, el ***contrarrecíproco*** o ***contrapositiva*** de una proposición condicional es el resultado de cambiar los papeles de las variables que intervienen en la proposición y negar cada una de ellas, es decir, para $p \rightarrow q$ el contrarrecíproco es $\boldsymbol{\neg q \rightarrow \neg p}$. La proposición $\boldsymbol{\neg p \rightarrow \neg q}$ es la ***inversa*** de $p \rightarrow q$.

La contrarrecíproca $\boldsymbol{\neg q \rightarrow \neg p}$ de una implicación $p \rightarrow q$ tiene la misma tabla de verdad que $p \rightarrow q$. Para verlo, ten en cuenta que la contrarrecíproca es falsa sólo cuando $\boldsymbol{\neg p}$ es falsa y $\boldsymbol{\neg q}$ es verdadera, esto es, sólo cuando *p* es verdadera y *q* falsa. Cuando dos fórmulas tienen siempre los mismos valores de verdad las llamamos equivalentes, de tal forma que una implicación y su contrarrecíproca son equivalentes.

Uno de los errores más comunes en lógica es suponer que la recíproca o la inversa son equivalentes a la implicación directa.

© Todos los derechos reservados

EJEMPLO 1.5.9.

¿Cuáles son las contrarrecíproca, recíproca e inversa de la implicación «El equipo local gana siempre que llueve»?

Solución:

Como «q siempre que p» es una forma de expresar la implicación $p \rightarrow q$, la afirmaciónoriginal se puede reescribir como:

«Si llueve, entonces el equipo local gana»

Consecuentemente, la contrarrecíproca de esta implicación es:

«Si el equipo local no gana, no llueve».

La recíproca es:

«Si el equipo local gana, entonces llueve».

La inversa es:

«Si no llueve, entonces el equipo local no ganan».

Sólo el contrarrecíproco es equivalente a la afirmación original.

© TODOS LOS DERECHOS RESERVADOS

- **Bicondicional:** toma valor 1 si ambas proposiciones tienen el mismo valor de verdad, en caso contrario su valor es 0.

  Definición 1.5.7: Sean p y q proposiciones. La proposición "***p si y solo si q***", denominada ***bicondicional*** o ***doble implicación*** de p y q, y denotada como $p \leftrightarrow q$, es la proposición que es verdadera cuando p y q tienen los mismos valores de verdad y falsa en los otros casos.

  Observa que la doble implicación es verdadera precisamente cuando las implicaciones $p \rightarrow q$ y $q \rightarrow p$ son verdaderas. Debido a esto, la terminología "***p si y solo si q***" se usa para esta bicondicional y simbólicamente se escribe combinando los símbolos $\rightarrow$ y $\leftarrow$. Hay otras formas en las que comúnmente se expresa $p \leftrightarrow q$:

  – p es necesario y suficiente para q.
  – p si y solo si q.
  – p es equivalente a decir q.
  – Si p, entonces q, y viceversa.
  – Si p, entonces q, y recíprocamente.

Tabla 1.9: Tabla de verdad del operador Bicondicional.

p	q	$p \leftrightarrow q$
1	1	1
1	0	0
0	1	0
0	0	1

Observa que $p \leftrightarrow q$ tiene exactamente los mismos valores de verdad que $(p \rightarrow q) \wedge (p \rightarrow q)$, por lo que se dice que ambas proposiciones son equivalentes.

EJEMPLO 1.5.10.

Sean las proposiciones:

p: programo en el lenguaje C#.

q: mi PC tiene sistema operativo Windows 10.

Por tal motivo:

$p \leftrightarrow q$: programo en el lenguaje C# si y solo si mi PC tiene sistema operativo Windows 10.

Otra manera de expresarlo es:

$p \leftrightarrow q$: que programe en el lenguaje C# es necesario y suficiente para que mi PC tenga sistema operativo Windows 10.

Por tal motivo:

$p \leftrightarrow q$: programo en el lenguaje C# si y solo si mi PC tiene sistema operativo Windows 10.

Otra manera de expresarlo es:

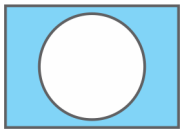
$p \leftrightarrow q$: que programe en el lenguaje C# es necesario y suficiente para que mi PC tenga sistema operativo Windows 10.

En Matemática I se estudiaron las operaciones entre conjuntos, las cuáles podíamos visualizar a través de los diagramas de Venn. A continuación se muestra (figura 1.1) la estrecha relación que poseen las operaciones entre conjuntos y lo operadores lógicos.

© TODOS LOS DERECHOS RESERVADOS

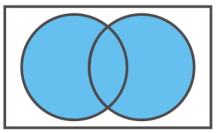
¬p se asemeja a A^C

p	*¬p*
A	A^C
0	1
1	0



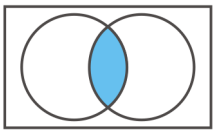
p ∨ q se asemeja a A ∪ B

p	*q*	
A	*B*	
0	0	0
0	1	1
1	0	1
1	1	1



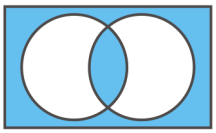
p ∧ q se asemeja A ∩ B

p	*q*	
A	*B*	
0	0	0
0	1	0
1	0	0
1	1	1



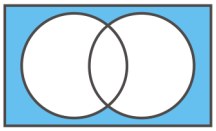
p ⇔ q se asemeja a $[A \Delta B]^C$

p	*q*	*p ∨ q*
A	*B*	$[A \Delta B]^C$
0	0	1
0	1	0
1	0	0
1	1	1



p ⊕ q se asemeja a A Δ B

p	*q*	*p ⊕ q*
A	*B*	*A Δ B*
0	0	0
0	1	1
1	0	1
1	1	0



p ⇒ q se asemeja a $[A \setminus B]^C$

p	*q*	*p ⇒ q*
A	*B*	*[A \ B]C*
0	0	1
0	1	1
1	0	0
1	1	1

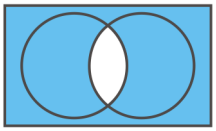


Figura 1.1. Relación entre las operaciones con conjuntos y los operadores de la lógica proposicional.

1.5.2. Traducción de expresiones del lenguaje natural

Existen muchas razones para traducir expresiones del lenguaje natural a expresiones con variables proposicionales y conectivos lógicos. Todos los lenguajes del ser humano son a menudo ambiguos. Trasladar frases a expresiones lógicas trae consigo evitar estas ambigüedades. Tener en cuenta que puede que esto conlleve hacer un conjunto de suposiciones razonables basadas en el sentido que se le dé a la frase. Por otra parte, una vez que hemos traducido frases del lenguaje natural a expresiones lógicas, podemos analizar estas expresiones lógicas para determinar sus valores de verdad, las podemos manipular para razonar sobre ellas. El paso del lenguaje natural al lenguaje formal se conoce como ***formalización***. Para ilustrar el proceso de formalizar, consideremos los siguientes ejemplos:

© Todos los derechos reservados

EJEMPLO 1.5.11.

Formalice la siguiente frase: *Puedes acceder a Internet desde el campus sólo si estudias Informática o no eres alumno de primero*.

Solución:

Hay muchas formas de formalizar esta frase. Aunque es posible representar la frase mediante una variable proposicional simple, como p, no sería útil para analizar su significado o razonar con ella. Así, utilizaremos variables proposicionales para representar cada parte de la oración y determinar los conectivos lógicos apropiados entre ellas. En particular, representaremos las frases como sigue:

p: Puedes acceder a Internet desde el campus.

q: Estudias Informática.

r: Eres alumno de primero.

Considerando que «sólo si» es una forma de expresar una implicación, la frase se puede representar como

$p \rightarrow (q \vee \neg r)$.

EJEMPLO 1.5.12.

¿Cómo se puede formalizar la siguiente frase?:

No puedes montar en la montaña rusa si mides menos de 1,20 metros, a no ser que seas mayor de dieciséis años.

Solución:

De nuevo, hay muchas formas de formalizar esta frase. En particular, si representamos por:

q: Puedes montar en la montaña rusa.

r: Mides menos de 1,20 metros.

s: Eres mayor de dieciséis años.

$(r \wedge s) \rightarrow \neg q$.

© TODOS LOS DERECHOS RESERVADOS

1.5.2.1. Especificaciones de sistema

Traducir oraciones del lenguaje natural, como el español, a expresiones lógicas es una parte esencial de la especificación tanto de sistemas hardware como software. Los ingenieros de software y de sistemas reciben los requerimientos en lenguaje natural y producen especificaciones precisas y sin ambigüedades que pueden usarse como base para desarrollo de sistemas. El siguiente ejemplo muestra cómo se pueden utilizar las variables proposicionales en este proceso.

EJEMPLO 1.5.13.

Expresa la especificación: *la respuesta automatizada no se puede enviar cuando el sistema de archivos está lleno*.

Solución: una forma posible de traducir esto es denotar:

p: La respuesta automatizada se puede enviar.

q: El sistema de archivos está lleno.

Entonces, $\neg p$ representa a «No se cumple que la respuesta automatizada se pueda enviar», lo que se puede expresar como «La respuesta automatizada no se puede enviar». Consecuentemente, nuestra especificación se puede representar mediante la implicación

$q \rightarrow \neg p$.

Las especificaciones de sistema no deberían contener requerimientos que puedan entrar en conflicto. Si así fuese, no habría forma de desarrollar un sistema que cumpliese todas las especificaciones. Consecuentemente, las expresiones proposicionales que representan esas especificaciones necesitan ser ***consistentes***. Para verificar esto, debe haber una asignación de valores de verdad a las variables de las expresiones que haga a todas las expresiones verdaderas.

EJEMPLO 1.5.14.

Determina si estas especificaciones de sistemas son consistentes:

- El mensaje de error se almacena en un buffer o se vuelve a transmitir.
- El mensaje de error no se almacena en el buffer.
- Si el mensaje de error se almacena en el buffer, entonces se vuelve a transmitir.

© TODOS LOS DERECHOS RESERVADOS

Solución:

Para determinar si estas expresiones son consistentes, primero las expresamos usando variables proposicionales:

p: El mensaje de error se almacena en un buffer.

q: El mensaje de error se vuelve a transmitir.

Las especificaciones se pueden escribir entonces como:

$p \vee q$, $\neg p$ y $p \rightarrow q$

Una asignación de valores de verdad que haga a las tres especificaciones verdaderas debe hacer *p* falsa para hacer $\neg p$ verdadera. Como queremos que $p \vee q$ sea verdadera, pero *p* debe ser falsa, *q* debe ser verdadera. Como $p \rightarrow q$ es verdadera cuando *p* es falsa y *q* verdadera, concluimos que *estas especificaciones son consistentes*, ya que las tres son verdaderas cuando *p* es falsa (0) y *q* verdadera (1). Podríamos haber llegado a la misma conclusión usando una tabla de verdad para examinar las cuatro posibles asignaciones de valores de verdad a *p* y *q*.

EJEMPLO 1.5.15.

¿Siguen siendo consistentes las especificaciones de sistema del ejemplo 1.5.14 si se añade la especificación «El mensaje de error no se vuelve a transmitir»?

Solución:

Por los razonamientos del ejemplo anterior, las tres especificaciones de ese ejemplo son verdaderas solo en el caso de que *p* es falsa (0) y *q* verdadera (1). Sin embargo, esta nueva especificación es $\neg q$, que es falsa cuando *q* es verdadera. Consecuentemente, *estas cuatro especificaciones son inconsistentes*.

1.5.3. INTERPRETACIÓN DE FÓRMULAS

Para interpretar una fórmula se construye una tabla de verdad asociada con todas las posibles interpretaciones de las variables proposicionales que conforman la fórmula y el valor de verdad de la fórmula completa para cada interpretación (asignación de valores veritativos a las variables). Una fórmula de *n* variables tendrá una tabla de verdad con 2^n fila, cada una constituye una interpretación.

© TODOS LOS DERECHOS RESERVADOS

Definición 1.5.8: A la hora de interpretar una fórmula se seguirá el siguiente orden para evaluar las operaciones que la componen:

1. Negación de variables o constantes.
2. Fórmulas entre agrupadores.
3. Negación de fórmulas.
4. Conjunciones.
5. Disyunciones.
6. Implicaciones.
7. Bicondicionales.

A este orden se le denomina precedencia de los operadores lógicos.

Ejemplo 1.5.16. Interpretación de fórmulas

A partir de la interpretación, ***p***: 1, ***q***: 0, ***r***: 1, determine el valor veritativo de las fórmulas siguientes:

a) $p \leftrightarrow (r \vee q)$

b) $\neg(p \wedge q) \vee (\neg p \wedge r)$

c) $(p \wedge \neg q) \rightarrow (q \vee \neg r)$

Solución:

a) $p \leftrightarrow (r \vee q)$

Primero se determina el valor de verdad de la proposición $(r \vee q)$ ya que los paréntesis le dan prioridad sobre el operador $\leftrightarrow$. El operador $\vee$ establece que la proposición es falsa solo cuando ambas variables son 0, lo cual no se cumple en este caso, por tanto $(r \vee q)$: 1. Luego se analiza la bicondicional con el resultado obtenido anteriormente $p \leftrightarrow 1$, la tabla de verdad de este operador plantea que si ambas variables tienen igual valor de verdad entonces la proposición es verdadera, como p: 1 entonces $p \leftrightarrow 1$ toma valor 1, es decir, la fórmula $p \leftrightarrow (r \vee q)$ es verdadera.

b) $\neg(p \wedge q) \vee (\neg p \wedge r)$

En este caso se debe analizar por separado las expresiones $\neg(p \wedge q)$ y $(\neg p \wedge r)$, la primera de ellas tiene valor 0 para lo cual se siguen los pasos siguientes: q es 0 por tanto $\neg q$ es 1, luego utilizando la tabla de verdad del operador $\wedge$ se obtiene que $(p \wedge \neg q)$ es 1 y por ende $\neg(p \wedge \neg q)$ es 0. Por la otra parte se tiene que p es 0, entonces $(\neg p \wedge r)$ es 0. Finalmente uniendo el resultado de ambas expresiones se obtiene $(0 \vee 0)$ lo cual es 0, por tanto $\neg(p \wedge \neg q) \vee (\neg p \wedge r)$ es falsa para la interpretación dada.

© Todos los derechos reservados

c) $(p \wedge \neg q) \rightarrow (q \vee \neg r)$

Según se analizó en el inciso anterior $(p \wedge \neg q)$ es 1, mientras que $(q \vee \neg r)$ es 0 ya que q y $\neg r$ son 0. En último lugar $1 \rightarrow 0$ es falso, por tanto la fórmula tiene valor 0, es falsa.

EJEMPLO 1.5.17. INTERPRETACIÓN DE FÓRMULAS

Elabore la tabla de verdad con cada una de las interpretaciones de las fórmulas siguientes:

a) $(p \wedge \neg q) \rightarrow r$

b) $\neg(p \wedge q) \rightarrow (\neg p \wedge \neg q)$

c) $(p \rightarrow q) \leftrightarrow ((p \vee r) \wedge s)$

Solución:

a) $(p) \rightarrow r$

Para determinar los valores de verdad de la fórmula planteada lo primero es construir una tabla donde las columnas estén formadas por las proposiciones que conforman la fórmula, desde las más simples a la más compleja, en el orden de los pasos seguidos en su construcción. Como se puede ver en la tabla 1.10 inicialmente se establecen los posibles valores de verdad de las primeras variables proposicionales del enunciado.

Tabla 1.10: Tabla de verdad del ejemplo 1.5.2 inciso a, paso 1.

p	q	$\neg q$	$p \wedge \neg q$	r	$(p \wedge \neg q) \rightarrow r$
1	1				
1	0				
0	1				
0	0				

Posteriormente se rellena el resto de las columnas (tabla 1.11), apoyándose en las tablas de verdad de los conectivos lógicos.

© TODOS LOS DERECHOS RESERVADOS

Tabla 1.11: Tabla de verdad del ejemplo 1.5.17 inciso a, paso 2.

p	q	$\neg q$	$p \wedge \neg q$	r	$(p \wedge \neg q) \rightarrow r$
1	1	0	0		
1	0	1	1		
0	1	0	0		
0	0	1	0		

Cuando se introduce una nueva variable proposicional en la tabla, se deben duplicar los valores de verdad obtenidos para combinarlos con los dos valores de verdad posibles de la nueva variable proposicional, en este caso la variable r, señalados en rojo y verde en la tabla 1.12.

Tabla 1.12: Tabla de verdad del ejemplo 1.5.17 inciso a, paso final.

p	q	$\neg q$	$p \wedge \neg q$	r	$(p \wedge \neg q) \rightarrow r$
1	1	0	0	1	**1**
1	0	1	1	1	**1**
0	1	0	0	1	**1**
0	0	1	0	1	**1**
1	1	0	0	0	**1**
1	0	1	1	0	**0**
0	1	0	0	0	**1**
0	0	1	0	0	**1**

Finalmente los valores de verdad de la fórmula $((p \wedge \neg q) \rightarrow r)$ para cada interpretación son los señalados en la última columna de la tabla 1.13. Una vez explicado el procedimiento paso a paso, en las siguientes interpretaciones de fórmulas se elaborará la tabla de manera íntegra, en un solo paso.

b) $(\overline{p \vee q}) \rightarrow (\overline{p} \wedge \overline{q})$

Tabla 1.13: Tabla de verdad del ejemplo 1.5.2 inciso b.

p	q	$p \vee q$	$\overline{p \vee q}$	$\overline{p}$	$\overline{q}$	$\overline{p} \wedge \overline{q}$	$(\overline{p \vee q}) \rightarrow (\overline{p} \wedge \overline{q})$
1	1	1	**0**	0	0	**0**	**1**
1	0	1	**0**	0	1	**0**	**1**
0	1	1	**0**	1	0	**0**	**1**
0	0	0	**1**	1	1	**1**	**1**

© Todos los derechos reservados

c) $(p \rightarrow q) \leftrightarrow ((p \vee r) \wedge s)$

En la tabla 1.14 la línea horizontal verde marca el duplicado de los valores de verdad obtenidos hasta el momento de introducir la variable *r* para combinarlos con sus dos valores de verdad (1 y 0) señalados en verde. Por otra parte línea horizontal roja en el medio de la tabla indica el duplicado resultante de incorporar la variable *s*.

Tabla 1.14: Tabla de verdad del ejemplo 1.5.2 inciso c.

p	q	$p \rightarrow q$	r	$p \vee r$	s	$(p \vee r) \wedge s$	$(p \rightarrow q) \leftrightarrow ((p \vee r) \wedge s)$
1	1	**1**	1	1	*1*	***1***	***1***
1	0	**0**	1	1	*1*	***1***	***0***
0	1	**1**	1	1	*1*	***1***	***1***
0	0	**1**	1	1	*1*	***1***	***1***
1	1	**1**	0	1	*1*	***1***	***1***
1	0	**0**	0	1	*1*	***1***	***0***
0	1	**1**	0	0	*1*	***0***	***0***
0	0	**1**	0	0	*1*	***0***	***0***
1	1	**1**	1	1	*0*	***0***	***0***
1	0	**0**	1	1	*0*	***0***	***1***
0	1	**1**	1	1	*0*	***0***	***0***
0	0	**1**	1	1	*0*	***0***	***0***
1	1	**1**	0	1	*0*	***0***	***0***
1	0	**0**	0	1	*0*	***0***	***1***
0	1	**1**	0	0	*0*	***0***	***0***
0	0	**1**	0	0	*0*	***0***	***0***

1.5.4. Clasificación de fórmulas

Definición 1.5.9: una fórmula *A* es ***modelo (verdadera)*** para una interpretación si y sólo si A es 1 para dicha interpretación. De lo contrario, A es un ***contramodelo (falsa)*** para dicha interpretación.

Definición 1.5.10: una fórmula *A* es una ***tautología*** si y sólo si todas sus interpretaciones son modelos.

© Todos los derechos reservados

Una fórmula *A* es una ***contradicción*** si y sólo si todas sus interpretaciones son contramodelos.

Una fórmula *A* es una ***contingencia*** si y sólo si posee modelos y contramodelos.

Es decir, en la tabla de una tautología la última columna, correspondiente a la proposición compuesta, está formada solo por 1. El inciso *b* del ejemplo 1.5.2 constituye una muestra de tautología. Podemos usar las ideas de tautología e implicación para describir lo que entendemos por un argumento válido. Esto tendrá interés a primordial a la hora de desarrollar la capacidad necesaria para demostrar teoremas matemáticos.

Por el contrario una fórmula que tenga valor 0 (falso) para cualquier interpretación, recibe el nombre de ***contradicción***. La tabla 1.1.5 muestra un ejemplo de contradicción.

Tabla 1.15: Ejemplo de contradicción.

p	$\neg p$	$p \wedge \neg p$
1	0	**0**
0	1	**0**

Aquellas proposiciones compuestas que cumplen que entre sus interpretaciones existen valores verdaderos y falsos, se denominan ***contingencias***. Los incisos a y c del ejemplo 1.5.2 son muestra de contingencias.

Se dice que una fórmula es ***satisfacible*** si es verdadera en al menos una de sus interpretaciones, en caso contrario, es decir, si es falsa para todas sus interpretaciones se dice que es una fórmula ***insatisfacible***. Las tautologías y las contingencias constituyen fórmulas satisfacibles mientras que las contradicciones son fórmulas insatisfacibles.

EJEMPLO 1.5.18.

Clasifique la siguiente fórmula en tautología, contradicción o contingencia:

$$A: \neg p \wedge p \vee [q \vee \neg q]$$

Para ello partimos de realizar la tabla de verdad. Veamos:

© TODOS LOS DERECHOS RESERVADOS

				A	B	
p	***q***	***¬p***	***¬q***	***¬p ∧ p***	***q ∨ ¬q***	***A ∨ B***
0	**0**	1	1	0	1	1
0	**1**	1	0	0	1	1
1	**0**	0	1	0	1	1
1	**1**	0	0	0	1	1

Como todas las interpretaciones son modelos entonces la fórmula *A* es una tautología.

1.6. DIAGRAMA DE DECISIÓN BINARIO (DDB)

El problema de decidir si una fórmula en la lógica proposicional será satisfactoria desde el punto de vista semántico tiene muchas e importantes aplicaciones en la ciencia de la computación.

Los Diagramas de Decisión Binaria (DDB) conocidos en idioma ingles por "*Binary Decision Diagram* (BDD)" s una estructura de datos utilizada para representar una semánticamente las formulas de la lógica proposicional. En un DDB las formulas son representadas por un grafo dirigido que algunas bibliografías son llamados grafo acíclico dirigido proposicional (GADP).

La idea básica de donde proviene esta estructura de datos es la expansión de Shannon. Una función booleana se divide en dos sub-funciones (cofactores) mediante la incorporación de una variable (cf. forma normal **if-then-else**). Si tal sub-función se considera como un sub-árbol, entonces se puede representar por un árbol de decisión binario. Los diagramas de decisión binarios fueron introducidos por Lee[1] , y más adelante estudiados y dados a conocer mejor por Akers[2] y Boute[3] .

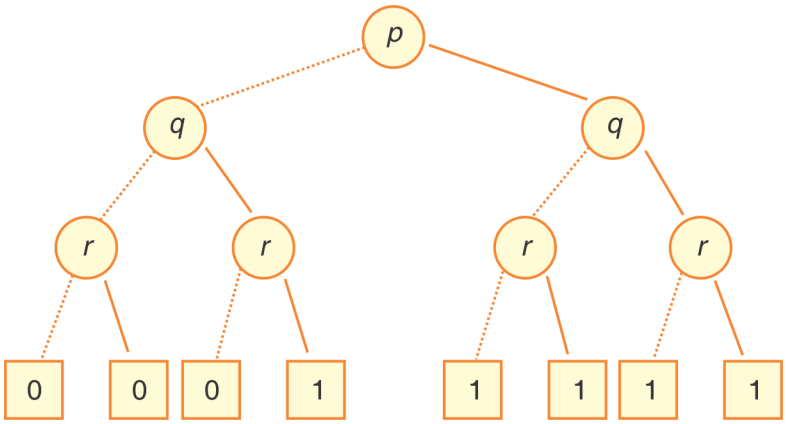
Definición 1.6.1: un diagrama de decisión binario es un grafo acíclico dirigido con uno o dos nodos terminales etiquetados con 0 (Falso) o 1 (Verdadero) y un conjunto de nodos variables con dos hijos cada nodo variable está etiquetado con un símbolo proposicional.

© Todos los derechos reservados

1. C. Y. Lee. "Representation of Switching Circuits by Binary-Decision Programs". Bell Systems Technical Journal, 38:985-999, 1959.
2. Sheldon B. Akers. Binary Decision Diagrams, IEEE Transactions on Computers, C-27(6):509-516, June 1978.
3. Raymond T. Boute, "The Binary Decision Machine as a programmable controller". EUROMICRO Newsletter, Vol. 1(2):16-22, January 1976.

EJEMPLO 1.6.1.

En la siguiente figura aparece el DDB de la fórmula $p \vee (q \wedge r)$.



Es posible modificar la estructura de un DDB para poder obtener una representación más reducida y consistente sin perder la capacidad de dicho diagrama para poder manipular todas las posibles interpretaciones de la fórmula que está modelando. Estas modificaciones son conocidas como reducciones de tal forma que luego de efectuarlas el DDB no se puede reducir más.

Definición 1.6.2: un DDB está reducido (DDBR) si cumple con:

- Unicidad: no existen dos nodos en el diagrama etiquetados con el mismo símbolo proposicional y con los mismos hijos izquierdo y derecho (no existen nodos duplicados).
- No-redundancia: no existen nodos variables con hijos izquierdo y derecho idénticos (nodos redundantes).

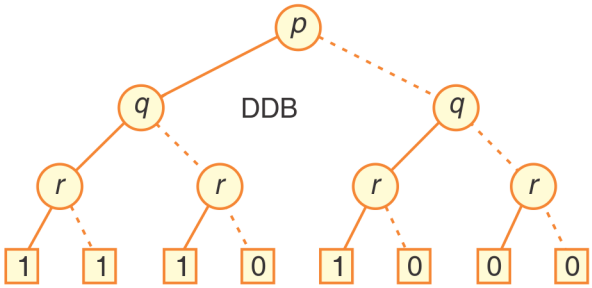
EJEMPLO 1.6.2.

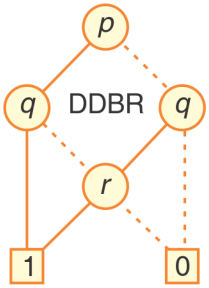
Para cada uno de las expresiones obtenga el DDB y el DDBR.

a) $(p \wedge q) \vee (p \wedge r) \vee (q \wedge r)$.

© TODOS LOS DERECHOS RESERVADOS

p	q	r	$(p \wedge q) = A$	$(p \wedge r) = B$	$(q \wedge r) = C$	$A \vee B \vee C$
1	1	1	1	1	1	1
1	1	0	1	0	0	1
1	0	1	0	1	0	1
1	0	0	0	0	0	0
0	1	1	0	0	1	1
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

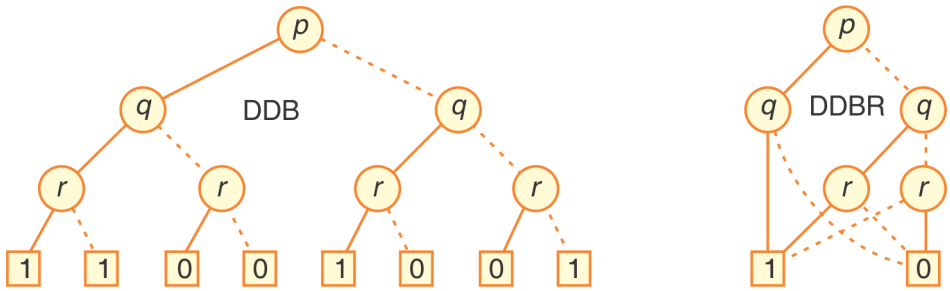




b) $(\neg p \wedge \neg q \wedge \neg r) \vee (p \wedge q) \vee (q \wedge r)$.

p	q	r	$(\neg p \wedge \neg q \wedge \neg r) = A$	$(p \wedge q) = B$	$(q \wedge r) = C$	$A \vee B \vee C$
1	1	1	0	1	1	1
1	1	0	0	1	0	1
1	0	1	0	0	0	0
1	0	0	0	0	0	0
0	1	1	0	0	1	1
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	1	0	0	1

© TODOS LOS DERECHOS RESERVADOS



Definición 1.6.3: un DDB está ordenado (DDBO) si existe un orden entre los símbolos proposicionales $x_1 < x_2 < \ldots < x_n$ tal que para todo nodo variable etiquetado con x_i, si uno de sus nodos hijos es un nodo variable etiquetado con x_j entonces $x_i < x_j$

Ejemplo 1.6.3.

Si $F \equiv (x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2)$ y $x_1 < y_1 < x_2 < y_2$. Su DDBO sería:

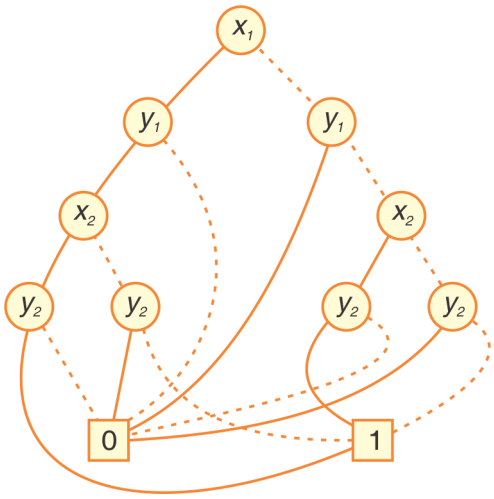


Figura 1.2. DDBO.

Una clasificación importante de los DDB son los que cumplen ser reducidos y ordenados (DDBRO) los cuales cumplen las siguientes propiedades (una vez fijado en orden entre las variables):

- Toda fórmula es equivalente a un único DDBRO.
- Una fórmula es válida si su DDBRO equivalente está formado por el nodo terminal 1.

© Todos los derechos reservados

- Una fórmula es insatisfacible si su DDBRO equivalente está formado por el nodo terminal 0.
- Los modelos de una fórmula se pueden determinar a partir de los caminos en su DDBRO equivalente que van desde el nodo raíz hasta el nodo terminal 1.

EJEMPLO 1.6.4.

El DDBRO equivalente a la fórmula del ejemplo 1.6.3 es

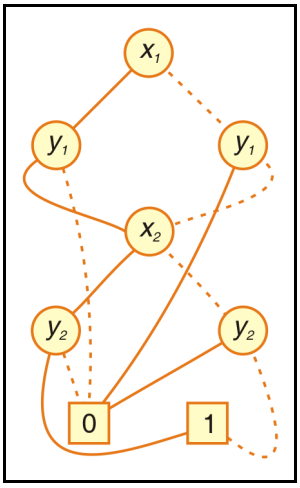


Figura 1.3. DDBRO

© TODOS LOS DERECHOS RESERVADOS

1.7. BÚSQUEDAS BOOLEANAS

Los conectivos lógicos tienen un amplio campo de aplicación en las búsquedas en grandes colecciones de información como, por ejemplo, los índices de páginas web. Como estas búsquedas emplean técnicas de lógica proposicional, se denominan ***búsquedas booleanas***.

En las búsquedas booleanas se usa la conexión *AND* para emparejar datos almacenados que contengan los dos términos de búsqueda, la conexión *OR* se usa para emparejar uno o ambos términos de la búsqueda y la conexión *NOT* (a veces escrita *AND NOT*) se usa para excluir un término particular de búsqueda. Cuando se utilizan búsquedas booleanas para localizar informaci6n de potencial interés, se requiere con frecuencia una planificación detallada de cómo emplear los conectivos. El ejemplo 1.7.1. ilustra cómo llevar a cabo búsquedas booleanas.

EJEMPLO 1.7.1.

Búsquedas en páginas web:

La mayoría del software de búsqueda en la web emplea técnicas de búsqueda booleana, las cuales nos pueden ayudar a encontrar páginas web sobre temas particulares. Por ejemplo, usando una búsqueda booleana para encontrar páginas web sobre una universidad en Madrid, podemos buscar páginas que concuerden con MADRID AND UNIVERSIDAD. El resultado de esta búsqueda incluirá aquellas páginas que contengan las dos palabras MADRID y UNIVERSIDAD. Incluirá todas las páginas de interés junto con otras relacionadas con los criterios de búsqueda. Posteriormente, para encontrar páginas que traten de una universidad en Madrid o Valencia, podemos buscar páginas que concuerden con (MADRID OR VALENCIA) AND UNIVERSIDAD. (Nota: Aquí el operador AND tiene precedencia sobre el operador OR). El resultado de esta búsqueda incluirá todas las páginas que contengan la palabra UNIVERSIDAD y bien las palabras MADRID o la palabra VALENCIA. De nuevo, aparecerán páginas no deseadas. Finalmente, para encontrar páginas web que traten de una universidad en York en Inglaterra (y no en NuevaYork), debemos mirar las páginas que concuerdan con YORK y UNIVERSIDAD, pero como el resultado incluirá páginas acerca de alguna universidad en Nueva York, así como en York, se debería buscar aquellas páginas que concuerden con (YORK AND UNIVERSIDAD) NOT NUEVA. El resultado de esta búsqueda incluye páginas que contienen tanto la palabra YORK como UNIVERSIDAD, pero no contienen la palabra NUEVA.

© TODOS LOS DERECHOS RESERVADOS

1.8. JUEGOS DE LÓGICA

Aquellos juegos que se pueden resolver usando el razonamiento lógico se conocen como juegos lógicos. Resolver juegos lógicos es una excelente forma de practicar con las reglas de la lógica. Hay programas de ordenador diseñados para desarrollar razonamiento lógico que a menudo utilizan juegos de lógica para ilustrar sus capacidades. Mucha gente se divierte resolviendo juegos de lógica que se publican en libros y revistas como actividad recreativa.

Discutiremos en este apartado dos juegos de lógica. Empezamos con uno que fue planteado inicialmente por Raymond Smullyan, un maestro de los juegos de lógica, que ha publicado más de una docena de libros con interesantes juegos relacionados con el razonamiento lógico.

EJEMPLO 1.8.1.

En (Smullyan, 1978) Smullyan planteó muchos juegos lógicos acerca de una isla con dos clases de habitantes: caballeros, que siempre dicen la verdad, y sus opuestos, villanos, que siempre mienten. Te encuentras a dos personas, *A* y *B*. ¿Qué son *A* y *B* si *A* dice «*B* es un caballero» y *B* dice «Los dos somos de clases opuestas»?

Solución:

Haremos un análisis lógico donde se establece que p y q serán las afirmaciones de que A es un caballero y B es un caballero, respectivamente, de tal forma que $\neg p$ y $\neg q$ son las afirmaciones de que A es un villano y B es un villano, respectivamente.

Consideramos primero la posibilidad de que A es un caballero; ésta es la afirmación de que p es verdadera. Si A es un caballero, entonces dice la verdad cuando dice que B es un caballero; por tanto, q es verdadera, y A y B son de la misma clase. Sin embargo, si B es un caballero, entonces la afirmación de B de que A y B son de clases opuestas, la afirmación $(p \wedge \neg q) \vee (\neg p \wedge q)$ tendría que ser verdadera, lo que no se cumple, porque A y B son ambos caballeros. Consecuentemente, podemos concluir que A no es un caballero, es decir, p es falsa.

Si A es un villano, como todo lo que dice es falso, la afirmación de A de que B es un caballero, es decir, que q es verdadera, es una mentira, lo que significa que q es falsa y B es también un villano. Además, si B es un villano, la afirmación de B de que A y B son de clases opuesta es una mentira, lo que es consistente con que tanto A como B sean villanos. Concluimos, por tanto, que A y B son villanos.

A continuación, planteamos un juego de lógica conocido como el ***juego de los chicos con barro*** para el caso de dos chicos.

EJEMPLO 1.8.2.

Un padre les dice a sus dos hijos, un chico y una chica, que jueguen en el jardín sin ensuciarse. Sin embargo, jugando, los dos se manchan la frente de barro. Cuando los chicos acaban de jugar, su padre dice «Al menos uno de vosotros se ha manchado la frente de barro» y entonces le pide a los chicos que respondan «Sí» o «No» a la pregunta: «¿Sabes si tienes la frente manchada de barro?» El padre hace la pregunta dos veces. ¿Qué responderán los chicos cada vez que el padre hace la pregunta suponiendo que un chico puede ver si su hermano o hermana se ha manchado la frente, pero no puede verse la suya? Suponemos que los chicos son honestos y que responden simultáneamente a cada pregunta.

Solución:

Sea p la afirmación de que el hijo se ha manchado la frente y sea q la afirmación de que la hija se ha manchado la frente. Cuando el padre dice que al menos uno de los dos chicos se ha manchado la frente está afirmando que la disyunción $p \vee q$ es verdadera. Ambos chicos responderán «No» la primera vez que se les hace la pregunta porque cada uno sólo ve barro en la frente del otro. Esto es, el hijo sabe que q es verdadera, pero no sabe si p es verdadera, y la hija sabe que p es verdadera, pero no sabe si q es verdadera.

Una vez que el hijo ha respondido «No» a la primera pregunta, la hija puede determinar que q debe ser verdadera. Esto es así porque cuando se hace la primera pregunta, el hijo sabe que $p \vee q$ es verdadera, pero no puede determinar si p es verdadera. Usando esta

© TODOS LOS DERECHOS RESERVADOS

información, la hija puede concluir que q debe ser verdadera, ya que si q fuese falsa, el hijo podría haber razonado que debido a que $p \vee q$ es verdadera, entonces p debe ser verdadera, y él habría respondido «Sí» a la primera pregunta. El hijo puede razonar de la misma forma para determinar que p debe ser verdadera. De aquí se sigue que la respuesta de ambos chicos es «Sí» a la segunda pregunta.

1.9. LÓGICA Y OPERACIONES CON BITS

Los ordenadores representan la información usando bits. Un ***bit*** tiene dos valores posibles: 0 (cero) y 1 (uno). El significado de la palabra bit viene de la expresión inglesa binary digit, ya que ceros y unos son los dígitos usados en las representaciones binarias de los números. El famoso estadístico John Tukey introdujo esta terminología en 1946. Un bit se puede utilizar para representar un valor de verdad, ya que dos son los valores de verdad: verdadero y falso. Como se suele hacer, usaremos el bit 1 para representar el valor verdadero (alto voltaje) y 0 para el falso (bajo voltaje); esto es, 1 representa V (verdadero) y 0 representa F (falso). Una variable se llama variable booleana si su valor es verdadero o falso. Por consiguiente, una variable booleana se puede representar usando un bit.

Las ***operaciones con bits*** en el ordenador se corresponden con los conectivos lógicos analizados en este capítulo. Reemplazando el valor verdadero por 1 y el valor falso por 0 en las tablas de verdad de los operadores $\wedge$, $\vee$ y $\oplus$, se obtienen las tablas para las correspondientes operaciones con bits. Utilizaremos las expresiones OR, AND y XOR para los operadores $\wedge$, $\vee$ y $\oplus$, respectivamente, como se hace en varios lenguajes de programación.

A menudo se representa información usando cadenas de bits, que son sucesiones de ceros y unos. En este caso, se pueden utilizar operaciones sobre cadenas de bits para manipular esta información.

Definición 1.9.1: una ***cadena de bits*** es una sucesión de cero o más bits. La longitud de esta cadena es el número de bits de la cadena.

Podemos extender las operaciones con bits a cadenas de bits. *Definimos las operaciones bit OR, AND y XOR de dos cadenas de la misma longitud como aquellas operaciones cuyo resultado es una nueva cadena cuyos bits son el resultado de aplicar las operaciones OR, AND y XOR a los correspondientes bits de cada una de las dos cadenas*. Usamos los símbolos $\wedge$, $\vee$ y $\oplus$ para representar las operaciones bits correspondientes.

Se ilustra el uso de estas operaciones con cadenas de bits en el siguiente ejemplo.

© Todos los derechos reservados

EJEMPLO 1.9.1.

Utilizar las operaciones con bits OR, AND y XOR a las cadenas 01 1011 0110 y 11 0001 1101.

Solución:

Los resultados de las operaciones bit OR, AND y XOR se obtienen aplicando los operadores OR, AND y XOR a los correspondientes bit. Esto nos da:

01 1011 0110

11 0001 1101

11 1011 1111 operación OR

01 0001 0100 operación AND

10 1010 1011 operación XOR

1.10. IMPLICACIÓN Y EQUIVALENCIA LÓGICA. LEYES DE LA LÓGICA PROPOSICIONAL

Un tipo importante de paso utilizado en argumentos matemáticos es la sustitución de una sentencia por otra de igual valor de verdad. Así, en la construcción de argumentos matemáticos se emplean con frecuencia métodos que producen proposiciones con el mismo valor de verdad que una fórmula dada. Resulta de especial interés para la lógica proposicional establecer relaciones entre fórmulas con características similares. Además de analizar tipos especiales de fórmulas que serán de gran interés para las aplicaciones de la lógica.

Las equivalencias lógicas son muy utilizadas tanto en las matemáticas como en la programación puesto que constituyen una herramienta en el proceso de modelación.

Se dice que dos fórmulas son equivalentes cuando, para un mismo orden de sus variables, ambas tienen interpretaciones idénticas. Veamos la definición formal con la que trabajaremos en este curso.

Dos fórmulas **A** y **B** son ***lógicamente equivalentes*** si para toda interpretación tienen el mismo valor de verdad, es decir, si la fórmula $\mathbf{A} \leftrightarrow \mathbf{B}$ es una tautología. La equivalencia lógica entre dos fórmulas **A** y **B** se denota como $\mathbf{A} \equiv \mathbf{B}$. En el inciso b del ejemplo 1.5.2 las fórmulas $\neg(p \vee q)$ y $(\neg p \wedge q)$ son lógicamente equivalentes ya que sus interpretaciones tienen el mismo valor de verdad (tabla 1.11). La equivalencia lógica cumple con las propiedades:

© TODOS LOS DERECHOS RESERVADOS

a) $\mathbf{A} \equiv \mathbf{A}$ (*reflexiva*).

b) Si $\mathbf{A} \equiv \mathbf{B}$, entonces $\mathbf{B} \equiv \mathbf{A}$ (*simétrica*).

c) Si $\mathbf{A} \equiv \mathbf{B}$ y $\mathbf{B} \equiv \mathbf{C}$, entonces $\mathbf{A} \equiv \mathbf{C}$ (*transitiva*).

Para dos fórmulas **A** y **B**, se dice que **A** ***implica lógicamente*** a **B** ($\mathbf{A} \Rightarrow \mathbf{B}$), si y solo si, no existen interpretaciones para las cuales **A** sea 1 y **B** sea 0, por tanto, la fórmula $\mathbf{A} \rightarrow \mathbf{B}$ es una tautología. En el inciso b del ejemplo 1.5.2 la fórmula $\neg(p \vee q)$ implica lógicamente a la fórmula $(\neg p \wedge \neg q)$ ya que para todas las interpretaciones de valor 1 de $\neg(p \vee q)$ la fórmula $(\neg p \wedge \neg q)$ toma valor 1 también (tabla 1.11). La implicación lógica cumple con las propiedades:

1. $\mathbf{A} \Rightarrow \mathbf{A}$ (*reflexiva*).
2. Si $\mathbf{A} \Rightarrow \mathbf{B}$ y $\mathbf{B} \Rightarrow \mathbf{A}$, entonces $\mathbf{A} \equiv \mathbf{B}$ (*antisimétrica*).
3. Si $\mathbf{A} \Rightarrow \mathbf{B}$ y $\mathbf{B} \Rightarrow \mathbf{C}$, entonces $\mathbf{A} \Rightarrow \mathbf{C}$ (*transitiva*).

EJEMPLO 1.10.1.

Implicación y equivalencia lógica.

Determine si son ciertas las afirmaciones siguientes:

a) $((p \rightarrow r) \wedge (q \rightarrow r))$ implica lógicamente a $((p \vee q) \rightarrow r)$

b) $(p \wedge (q \vee r))$ es lógicamente equivalente a $(p \vee (q \wedge \neg r))$

Solución:

a) Si $((p \rightarrow r) \wedge (q \rightarrow r)) \Rightarrow ((p \vee q) \rightarrow r)$ entonces $((p \rightarrow r) \wedge (q \rightarrow r)) \rightarrow ((p \vee q) \rightarrow r)$ es una tautología, para verificarlo se debe elaborar la tabla de verdad correspondiente (tabla 1.16).

Tabla 1.16: Comprobación de la implicación del ejemplo 1.6.1 inciso a.

p	r	$p \rightarrow r$	q	$(q \rightarrow r)$	$(p \rightarrow r) \wedge (q \rightarrow r)$	$p \vee q$	$(p \vee q) \rightarrow r$	$((p \rightarrow r) \wedge (q \rightarrow r)) \rightarrow ((p \vee q) \rightarrow r)$
1	1	1	1	1	**1**	1	**1**	**1**
0	1	1	1	1	**1**	1	**1**	**1**
1	0	0	1	0	**0**	1	**0**	**1**
0	0	1	1	0	**0**	1	**0**	**1**
1	1	1	0	1	**1**	1	**1**	**1**
0	1	1	0	1	**1**	0	**1**	**1**
1	0	0	0	1	**0**	1	**0**	**1**
0	0	1	0	1	**1**	0	**1**	**1**

© TODOS LOS DERECHOS RESERVADOS

Puesto que se obtuvo una tautología, queda demostrado que $((p \rightarrow r) \wedge (q \rightarrow r))$ implica lógicamente a $((p \vee q) \rightarrow r)$.

b) Si $(p \wedge (\neg q \vee r)) \equiv (p \vee (q \wedge \neg r))$ entonces $(p \wedge (\neg q \vee r)) \leftrightarrow (p \vee (q \wedge \neg r))$ es una tautología, para verificarlo se debe elaborar la tabla de verdad correspondiente (tabla 1.17).

Tabla 1.17: Comprobación de la implicación del ejemplo 1.6.1 inciso b.

q	$\neg q$	r	$\neg q \vee r$	p	$p \wedge (\neg q \vee r)$	$\neg r$	$q \vee \neg r$	$p \vee (q \wedge \neg r)$	$(p \wedge (\neg q \vee r)) \leftrightarrow (p \vee (q \wedge \neg r))$
1	0	1	1	1	**1**	0	0	**1**	**1**
0	1	1	1	1	**1**	0	0	**1**	**1**
1	0	0	0	1	**0**	1	1	**1**	**0**
0	1	0	1	1	**1**	1	0	**1**	**1**
1	0	1	1	0	**0**	0	0	**0**	**1**
0	1	1	1	0	**0**	0	0	**0**	**1**
1	0	0	0	0	**0**	1	1	**1**	**0**
0	1	0	1	0	**0**	1	0	**0**	**1**

Como se aprecia en la tabla 1.17 no se obtuvo una tautología, por tanto $(p \wedge (\neg q \vee r))$ y $(p \vee (q \wedge \neg r))$ no son lógicamente equivalentes.

A continuación se listan algunas de las leyes de la Lógica proposicional, donde **A**, **B** y **C** representan fórmulas cualesquiera, mientras que las constantes proposicionales 1 y 0 denotan tautologías y contradicciones respectivamente.

Leyes de la Lógica proposicional:

A partir de las relaciones definidas se da a continuación una lista de equivalencias e implicaciones lógicas importantes a las que se denominará de manera general leyes de la lógica proposicional. Las variables *A*, *B*, y *C* como siempre representan fórmulas cualesquiera, y las constantes proposicionales **1** y **0** denotan cualquier tautología y cualquier contradicción respectivamente.

1.	$\neg[\neg \mathbf{A}] \equiv \mathbf{A}$	(Ley de doble negación)
2.	$\mathbf{A} \vee \mathbf{A} \equiv \mathbf{A}$	(Ley de idempotencia)
3.	$\mathbf{A} \wedge \mathbf{A} \equiv \mathbf{A}$	
4.	$(\mathbf{A} \vee \mathbf{B}) \vee \mathbf{C} \equiv \mathbf{A} \vee (\mathbf{B} \vee \mathbf{C})$	(Ley asociativa)
5.	$(\mathbf{A} \wedge \mathbf{B}) \wedge \mathbf{C} \equiv \mathbf{A} \wedge (\mathbf{B} \wedge \mathbf{C})$	
6.	$\mathbf{A} \vee \mathbf{B} \equiv \mathbf{B} \vee \mathbf{A}$	(Ley conmutativa)
7.	$\mathbf{A} \wedge \mathbf{B} \equiv \mathbf{B} \wedge \mathbf{A}$	

© Todos los derechos reservados

8.	$\mathbf{A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)}$	(Ley distributiva)
9.	$\mathbf{A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)}$	
10.	$\mathbf{\neg[A \vee B] \equiv \neg A \wedge \neg B}$	(Ley de De Morgan)
11.	$\mathbf{\neg[A \wedge B] \equiv \neg A \vee \neg B}$	
12.	$\mathbf{A \vee 0 \equiv A}$	(Ley de identidad)
13.	$\mathbf{A \wedge 1 \equiv A}$	
14.	$\mathbf{A \vee 1 \equiv 1}$	(Ley de dominación)
15.	$\mathbf{A \wedge 0 \equiv 0}$	
16.	$\mathbf{A \wedge \neg A \equiv 0}$	(Ley de la contradicción)
17.	$\mathbf{A \vee \neg A \equiv 1}$	(Ley del tercero excluido)
18	$\mathbf{A \vee (A \wedge B) \equiv A}$	(Ley de absorción)
19	$\mathbf{A \wedge (A \vee B) \equiv A}$	
20	$\mathbf{A \rightarrow B \equiv \overline{A} \vee B}$	(Definición de la condicional)
21	$\mathbf{A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)}$	(Definición de la bicondicional)
22	$\mathbf{A \rightarrow B \equiv \overline{B} \rightarrow \overline{A}}$	(Ley del Contrarrecíproco o Transposición)

EJEMPLO 1.10.2.

Demuestre que las proposiciones $\neg[p \vee [\neg p \wedge q]]$ y $\neg p \wedge \neg q$ son lógicamente equivalentes.

Solución: podríamos utilizar una tabla de verdad para mostrar que estas fórmulas son equivalentes. En vez de ello, estableceremos la equivalencia desarrollando una serie de leyes de la lógica proposicional. Veamos:

$\neg[p \vee [p \wedge q]]$ $\equiv \neg p \wedge \neg[\neg p \wedge q]$ // por ley de De Morgan

$\equiv \neg p \wedge [\neg[\neg p] \vee \neg q]$ // por ley de De Morgan

$\equiv \neg p \wedge [p \vee \neg q]$ // por ley de la doble negación

$\equiv [\neg p \wedge p] \vee [\neg p \wedge \neg q]$ // por ley distributiva

$\equiv 0 \vee [\neg p \wedge \neg q]$ // por ley de la contradicción

$\equiv \neg p \wedge \neg q$ // por ley de identidad

© TODOS LOS DERECHOS RESERVADOS

EJEMPLO 1.10.3.

Demuestre que $[p \wedge q] \rightarrow [p \vee q]$ es una tautología.

Solución: para demostrar que esta sentencia es una tautología, usaremos equivalencias lógicas para demostrar que es lógicamente equivalente a **1**. (Nota: se podría haber hecho también mediante una tabla de la verdad).

$$
\begin{aligned}
[p \wedge q] \rightarrow [p \vee q] \quad & \equiv \neg[p \wedge q] \vee [p \wedge q] \text{ // por definición de la condicional} \\
& \equiv [\neg p \wedge \neg q] \vee [p \vee q] \text{ // por ley de De Morgan} \\
& \equiv [\neg p \vee p] \vee [\neg q \vee q] \text{ // por leyes asociativa y conmutativa} \\
& \equiv 1 \vee 1 \text{ // por ley del tercero excluido} \\
& \equiv 1 \text{ // por ley de idempotencia}
\end{aligned}
$$

Se puede usar una tabla de verdad para determinar si una fórmula es una tautología. Esta tabla se puede construir a mano para una proposición con un número reducido de variables, pero cuando el número de variables crece, el método manual se vuelve impracticable. Por ejemplo, hay más de 2^{20}= 1048 576 filas en la tabla de verdad de una proposición de veinte variables. Claramente, se necesita la ayuda de un ordenador si queremos determinar de esta forma si la fórmula de 20 variables es una tautología. Pero cuando hay mil variables, ¿puede un ordenador determinar en un plazo de tiempo razonable si una fórmula es una tautología? Revisar cada una de las 2^{1000} (un número con más de trescientas cifras) combinaciones de valores de verdad no puede hacerse en un ordenador ni en billones de años. Además, no se conoce ningún otro procedimiento que pueda seguir un ordenador para determinar en un plazo razonable de tiempo si una fórmula con un número tan grande de variables es una tautología.

© TODOS LOS DERECHOS RESERVADOS

EJEMPLO 1.10.3.

Para las proposiciones primitivas p, q, ¿existe una forma más sencilla de expresar la proposición compuesta $[p \vee q] \wedge \neg[\neg p \wedge q]$?; es decir, ¿podemos encontrar una proposición más sencilla que sea lógicamente equivalente a la dada?

Aquí vemos que:

$[p \vee q] \wedge \neg[\neg p \wedge q]$

$\equiv [p \vee q] \wedge [\neg[\neg p] \vee \neg q]$ // Ley de De Morgan

$\equiv [p \vee q] \wedge [p \vee \neg q]$ // Ley de la doble de negación

$\equiv p \vee [q \wedge \neg q]$ // Ley distributiva

$\equiv p \vee 0$ // Ley de la contradicción

$\equiv p$ // Ley de identidad

En consecuencia, tenemos que $[p \vee q] \wedge \neg[\neg p \wedge q] \equiv p$ de modo que podemos expresar la proposición compuesta DAD mediante la proposición lógicamente equivalente, más sencilla, p.

EJEMPLO 1.10.4.

Obtenga una expresión con un solo conectivo lógico equivalente a la proposición compuesta $\neg[\neg[[p \vee q] \wedge r] \vee \neg q]$, donde p, q, r son proposiciones primitivas.

Solución:

De las leyes de la lógica se sigue que:

$\neg[\neg[[p \vee q] \wedge r] \vee \neg q]$

$\equiv \neg\neg[[p \vee q] \wedge r] \wedge \neg[\neg q]$ // Ley de De Morgan.

$\equiv [[p \vee q] \wedge r] \wedge q$ // Ley de la doble de negación.

$\equiv [p \vee q] \wedge [r \wedge q]$ // Ley asociativa.

$\equiv [p \vee q] \wedge [q \wedge r]$ // Ley conmutativa.

$\equiv [[p \vee q] \wedge q] \wedge r$ // Ley asociativa.

$\equiv [q \wedge r]$ // Ley de absorción.

En consecuencia, la proposición original $\neg[\neg[[p \vee q] \wedge r] \vee \neg q]$ es lógicamente equivalente a la proposición $q \wedge r$ que sólo contiene dos proposiciones primitivas, ningún símbolo de negación y sólo una conectiva.

© TODOS LOS DERECHOS RESERVADOS

© Todos los derechos reservados

Resumen

© Todos los derechos reservados

© Todos los derechos reservados

Circuitos lógicos 2

Objetivos

- Definir los circuitos lógicos a partir de sus características fundamentales.
- Diseñar y representar circuitos lógicos utilizando las compuertas lógicas.
- Determinar la forma normal de un circuito lógico.
- Simplificar circuitos lógicos por el método algebraico y los mapas de Karnaugh.

© Todos los derechos reservados

2.1. INTRODUCCIÓN

La Lógica proposicional bivalente estudiada en el capítulo anterior ha sido de gran importancia en el desarrollo de la Teoría de Circuitos Lógicos, que constituye una de las bases para la construcción de dispositivos electrónicos de cómputo. Los circuitos digitales o lógicos operan de forma binaria, de manera que puede interpretarse una señal (de entrada o de salida) de bajo voltaje como 0 y una de alto voltaje como 1. En este contexto, 0 y 1 representan intervalos predefinidos de voltaje, aunque no todos los dispositivos se trabajan en función del voltaje, por ejemplo, en el caso de los conmutadores o interruptores los estados corresponden a la posición cerrada "ON" y abierta "OFF", pues no existe una posición intermedia. Esta cualidad de los circuitos lógicos permite utilizar el Álgebra de Boole para su análisis y diseño.

En este capítulo se trabaja con circuitos combinatorios en los cuales el valor de las salidas en cada instante se define de manera única para cada combinación de entradas. Estos circuitos se caracterizan por no tener memoria, las entradas previas y el estado del sistema no afectan la salida. Un circuito de estas características puede representarse gráficamente, mediante un diagrama de compuertas lógicas. En estos diagramas se representan las entradas, las salidas y las compuertas lógicas correspondientes a las operaciones y sus conexiones.

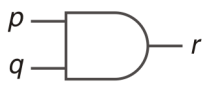
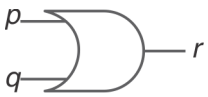
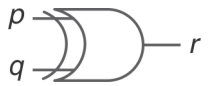
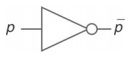
2.2. COMPUERTAS LÓGICAS. DISEÑO DE CIRCUITOS LÓGICOS

Las compuertas lógicas son dispositivos electrónicos que constituyen los circuitos lógicos fundamentales y tienen la capacidad de intercambiar los niveles de voltaje (bits). Las compuertas básicas son AND, OR y NOT; correspondientes a los operadores lógicos: conjunción, disyunción y negación, respectivamente. Las tres compuertas AND, OR y NOT son suficientes para diseñar un circuito lógico, sin embargo, existen otras compuertas lógicas complementarias como son NAND, NOR y XOR.

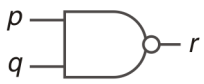
La compuerta NAND es la negación de la compuerta AND. El operador lógico que la identifica es ($\uparrow$) aunque se puede expresar a través de los operadores de negación y conjunción ($\wedge$). NOR representa la negación de la compuerta OR, se identifica con el operador ($\downarrow$) aunque se pueden utilizar los operadores de negación y disyunción ($\vee$).

La disyunción que se ha utilizado hasta este momento tiene un sentido inclusivo ya que acepta que ambas proposiciones sean verdaderas, es decir, ($p \vee q$) es 1 para p: 1 y q: 1. Existe también una disyunción exclusiva, que es la más utilizada en el lenguaje natural, y se representa con el operador ($\oplus$). En el caso de la disyunción exclusiva, ($p \oplus q$) solo es 1 si p: 1 o q: 1, pero no ambos simultáneamente. En el diseño de circuitos la disyunción exclusiva se representa a través de la compuerta XOR. La tabla 2.1 muestra un resumen con la representación y las operaciones lógicas correspondientes a las compuertas lógicas expuestas en este epígrafe.

Tabla 2.1: Compuertas lógicas.

Nombre	Representación	Operación lógica
AND		r: $p \wedge q$
OR		r: $p \vee q$ OR (inclusivo)
XOR		r: $p \oplus q$ OR (exclusivo)
NOT		p

© Todos los derechos reservados

Nombre	Representación	Operación lógica
NAND (NOT AND)		r: $p \uparrow q$ o r: $\overline{(p \wedge q)}$
NOR (NOT OR)		r: $p \downarrow q$ o r: $\overline{(p \vee q)}$

Con las compuertas básicas AND, OR y NOT se puede representar el operador condicional y el bicondicional, haciendo uso de las leyes de la lógica proposicional. Las figuras 2.1 y 2.2 muestran los circuitos resultantes.

**Nota*: según las leyes de equivalencia $(p \rightarrow q) \equiv \overline{p} \vee q$.

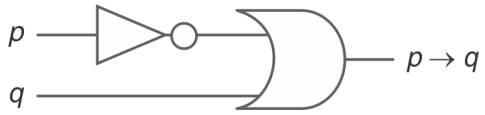


Figura 2.1. Circuito lógico del operador condicional.

**Nota*: según las leyes de equivalencia $(p \leftrightarrow q) \equiv (p \rightarrow q) \wedge (q \rightarrow p)$. Luego, como $(p \rightarrow q) \equiv \overline{p} \vee q$ y $(q \rightarrow p) \equiv \overline{q} \vee p$ entonces por transitividad $(p \leftrightarrow q) \equiv (\overline{p} \vee q) \wedge (\overline{q} \vee p)$.

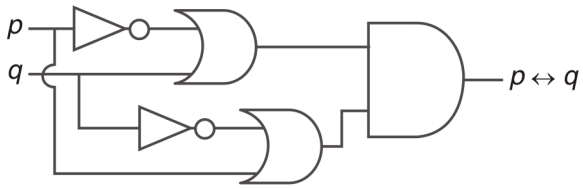


Figura 2.2. Circuito lógico del operador bicondicional.

Un recurso de gran utilidad en el diseño de circuitos lógicos es la **tabla lógica** que enumera los valores de salida para las diferentes combinaciones de valores de las entradas.

Ejemplo 2.2.1. Diseño de circuitos lógicos

Determine los circuitos lógicos y las tablas lógicas correspondientes a las siguientes proposiciones:

© Todos los derechos reservados

a) s: $p \uparrow (q \vee r)$

b) s: $(p \bar{\oplus} q) \wedge (r \downarrow q)$

Solución:

a) $p \uparrow (q \vee r)$

El circuito lógico correspondiente a la proposición del inciso a es el representado en la figura 2.3.

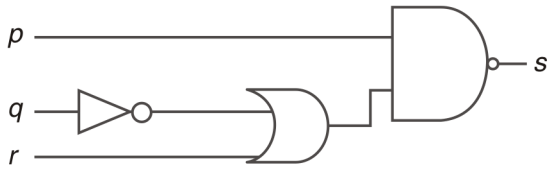


Figura 2.3. Circuito lógico del ejemplo 2.2.1 inciso a.

Los valores de la salida ***S*** correspondientes a las diferentes combinaciones de los valores de las entradas ***p***, ***q*** y ***r***, se recogen en la tabla 2.2. Como se muestra en el circuito, para obtener la salida ***S*** se debe negar los valores de la entrada ***q*** y luego negar la conjunción de ***p***, $\bar{q}$ y ***r***.

Tabla 2.2: Tabla lógica del ejemplo 2.2.1 inciso a.

p	q	r	s
1	1	1	**1**
0	1	1	**1**
1	0	1	**0**
0	0	1	**1**
1	1	0	**1**
0	1	0	**1**
1	0	0	**0**
0	0	0	**1**

b) s: $(\overline{p \bar{\oplus} \bar{q}}) \wedge (r \downarrow q)$

El circuito lógico correspondiente a la proposición del inciso *b* es el representado en la figura 2.4.

© Todos los derechos reservados

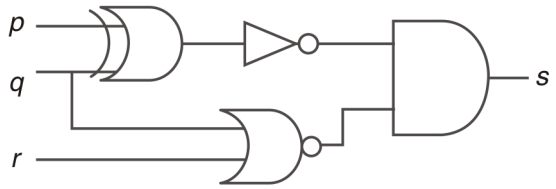


Figura 2.4. Circuito lógico del ejemplo 2.2.1 inciso b.

La tabla lógica correspondiente al circuito obtenido en el inciso b de ejemplo 2.2.1 se muestra a continuación:

Tabla 2.3: Tabla lógica del ejemplo 2.2.1 inciso b.

p	q	r	s
1	1	1	**0**
0	1	1	**0**
1	0	1	**0**
0	0	1	**0**
1	1	0	**0**
0	1	0	**0**
1	0	0	**0**
0	0	0	**1**

Ejemplo 2.2.2. Diseño de circuitos lógicos

Determine la proposición correspondiente a los circuitos lógicos representados y los valores de las salidas para las entradas ***x***: 1, ***y***: 0, ***z:*** 1.

a)

© Todos los derechos reservados

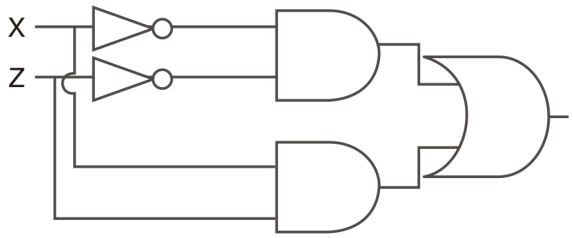


Figura 2.5. Circuito lógico del ejemplo 2.2.2 inciso a.

© Todos los derechos reservados

b)

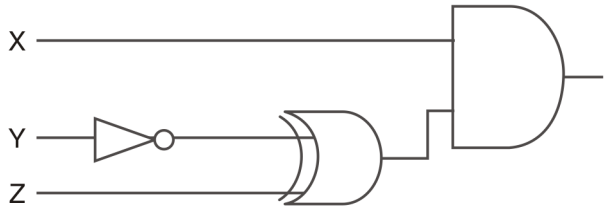


Figura 2.6. Circuito lógico del ejemplo 2.2.2 inciso b.

Solución:

a) $(\bar{x} \wedge \bar{z}) \vee (\bar{x} \wedge z)$

Para ***x: 1, y: 0, z: 1***

$(0 \wedge 0) \vee (1 \wedge 1)$

$(0 \vee 1) = 1$

b) $(\bar{y} \oplus z) \wedge x$

Para ***x: 1, y: 0, z: 1***

$(1 \oplus 1) \wedge 1$

$(0 \wedge 1) = 0$

© Todos los derechos reservados

2.3. FORMAS NORMALES

En el uso de la lógica en aplicaciones tecnológicas resulta muy útil la representación de fórmulas en su forma normal disyuntiva o en la forma normal conjuntiva.

Se dice que una fórmula está en **forma normal disyuntiva (FND)** si corresponde a una disyunción de cláusulas conjuntivas. Una **cláusula conjuntiva** es una conjunción de literales. En una misma cláusula no puede aparecer un literal y su complemento. Un **literal** es una fórmula compuesta por una variable proposicional (p) o su complemento, o sea, la negación de la variable proposicional (p). Por tanto una fórmula en FND tiene la forma:

$$A_1 \vee A_2 \vee \ldots \vee A_n$$

donde A_i ($1 \leq i \leq n$) es una conjunción de literales $l_1 \wedge l_2 \wedge \ldots \wedge l_n$.

Una fórmula está en **forma normal conjuntiva (FNC)** si es una conjunción de cláusulas disyuntivas. Una **cláusula disyuntiva** es una disyunción de literales. En una misma cláusula no puede aparecer un literal y su complemento. Una fórmula en FNC tiene la forma:

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

donde A_i ($1 \leq i \leq n$) es un disyunción de literales $l_1 \vee l_2 \vee \ldots \vee l_n$.

Todas las conjunciones de literales se reconocen como FNC y como FND porque pueden interpretarse como conjunciones de cláusulas de un solo literal o como disyunciones de una sola cláusula. Igualmente todas las disyunciones de literales están en FNC y como FND ya que pueden entenderse como disyunciones de cláusulas de un solo literal o como conjunciones de una sola cláusula.

EJEMPLO 2.3.1. FORMAS NORMALES

Identifique la forma normal correspondiente a las fórmulas siguientes:

a) $(\bar{x} \wedge \bar{z}) \vee (x \wedge z)$

b) $(p \vee \bar{r}) \wedge (\bar{p} \vee \bar{q} \vee r) \wedge (q \vee r)$

c) r

d) $p \wedge \bar{q}$

e) $\bar{p} \vee (r \wedge \bar{s})$

Solución:

a) $(\bar{x} \wedge \bar{z}) \vee (x \wedge z)$

Está en forma normal disyuntiva ya que las proposiciones $(\bar{x} \wedge \bar{z})$ y $(x \wedge z)$ están relacionadas por una disyunción y a su vez constituyen conjunciones de literales, por lo que cumplen con la forma $A_1 \vee A_2 \vee \ldots \vee A_n$, con A_i de la forma $l_1 \wedge l_2 \wedge \ldots \wedge l_n$.

b) $(p \vee \bar{r}) \wedge (\bar{p} \vee \bar{q} \vee r) \wedge (q \vee r)$

Está en forma normal conjuntiva ya que las proposiciones $(p \vee \bar{r})$, $(\bar{p} \vee \bar{q} \vee r)$ y $(q \vee r)$ están relacionadas por una conjunción y a su vez constituyen disyunciones de literales, por lo que cumplen con la forma $A_1 \wedge A_2 \wedge \ldots \wedge A_n$, con A_i de la forma $l_1 \vee l_2 \vee \ldots \vee l_n$.

c) r

Los literales se consideran ejemplos de forma normal conjuntiva y forma normal disyuntiva a la vez.

d) $p \wedge \bar{q}$

Está a la vez en forma normal conjuntiva y en forma normal disyuntiva.

e) $\bar{p} \vee (\bar{r} \wedge \bar{s})$

Está en forma normal disyuntiva ya que $(r \wedge \bar{s})$ es una conjunción de literales y la proposición p es un literal, que a su vez están relacionados por una disyunción, por tal razón cumplen con la forma $A_1 \vee A_2 \vee \ldots \vee A_n$, con A_i de la forma $l_1 \wedge l_2 \wedge \ldots \wedge l_n$.

© TODOS LOS DERECHOS RESERVADOS

Toda fórmula puede expresarse en forma normal conjuntiva o disyuntiva, para esto se aplican repetidamente un conjunto de las leyes de equivalencia lógica enumeradas en el epígrafe 1.6. Los pasos principales consisten en:

- Eliminar los operadores $\leftrightarrow$ utilizando la ley que define la bicondicional (Ley de equivalencia # 21).
- Eliminar los operadores $\rightarrow$ utilizando la ley que define la condicional (Ley de equivalencia # 20).
- Aplicar las leyes de Morgan para eliminar negaciones que no afecten únicamente a literales (Leyes de equivalencia # 10 y 11).
- Utilizar la Ley de doble negación para eliminar negaciones múltiples (Ley de equivalencia # 1).
- Aplicar las propiedades distributivas hasta obtener la forma normal deseada (Leyes de equivalencia # 8 y 9).

Luego de estos pasos se pueden aplicar otras leyes para simplificar la fórmula obtenida, por ejemplo:

- Ley de idempotencia (# 2 y 3) para eliminar literales repetidos.
- Ley inversa (# 16 y 17) para eliminar conjunciones o disyunciones entre un literal y su opuesto.
- Ley de absorción (# 18 y 19) erradica una conjunción incluida en una disyunción, o viceversa.

© Todos los derechos reservados

Ejemplo 2.3.2. Formas normales

Determine la FND de la expresión $(p \rightarrow \overline{q}) \rightarrow (\overline{p \wedge r})$

Solución:

Paso 1. Para eliminar el operador condicional principal que relaciona ambos miembros se utiliza la Ley de definición de la condicional, que es la # 20 en el listado de leyes presentado en el epígrafe 1.6.

$$(p \rightarrow q) \rightarrow (\overline{p \wedge r}) \equiv \overline{(p \rightarrow \overline{q})} \vee (\overline{p \wedge r})$$

Paso 2. Luego se elimina el operador condicional restante, utilizando la misma ley.

$$\overline{(p \rightarrow \overline{q})} \vee (\overline{p \wedge r}) \equiv (\overline{\overline{p}} \wedge \overline{\overline{q}}) \vee (\overline{p \wedge r})$$

Paso 3. Para erradicar las negaciones que no afectan a literales directamente se emplea las Leyes de De Morgan (# 10 y 11).

$$(\overline{\overline{p}} \wedge \overline{\overline{q}}) \vee (\overline{p \wedge r}) \equiv (\overline{\overline{p}} \wedge \overline{\overline{q}}) \vee (\overline{p} \vee \overline{r})$$

Paso 4. Las negaciones dobles desaparecen con la Ley de doble negación (# 1).

$(\overline{\overline{p}} \wedge \overline{\overline{q}}) \vee (\overline{p} \vee \overline{r}) \equiv (p \wedge q) \vee (\overline{p} \vee \overline{r})$

Paso 5. Luego aplicando las Leyes distributivas (# 8 y 9) se organiza convenientemente la expresión.

$(p \wedge q) \vee (\overline{p} \vee \overline{r}) \equiv (p \vee \overline{p}) \wedge (q \vee \overline{p}) \vee \overline{r}$

Paso 6. La Ley de tercero excluido (# 16) permite realizar la siguiente transformación.

$(p \vee \overline{p}) \wedge (q \vee \overline{p}) \vee \overline{r} \equiv 1 \wedge (q \vee \overline{p}) \vee \overline{r}$

Paso 7. Finalmente utilizando la Ley de identidad (# 13) se obtiene una expresión en FND.

$1 \wedge (q \vee \overline{p}) \vee \overline{r} \equiv q \vee \overline{p} \vee \overline{r}$

Ejemplo 2.3.3. Formas normales

Determine la FNC de la expresión $(p \wedge q) \vee \overline{(r \wedge \overline{s})}$

Solución:

Paso 1. Erradicar las negaciones que no afectan a literales directamente, Leyes de De Morgan (# 11).

$(p \wedge q) \vee \overline{(r \wedge \overline{s})} \equiv (p \wedge q) \vee (r \vee s)$

Paso 2. Con la Ley distributiva (# 8) se reorganiza la expresión de la siguiente forma.

$(p \wedge q) \vee (\overline{r} \vee s) \equiv (p \vee (\overline{r} \vee s)) \wedge (q \vee (\overline{r} \vee s))$

Paso 3. Por la Ley asociativa (# 4) se reorganiza la expresión de la siguiente forma.

$(p \wedge q) \vee (\overline{r} \vee s) \equiv (p \vee (\overline{r} \vee s)) \wedge (q \vee (\overline{r} \vee s))$

Finalmente la expresión $(p \vee \overline{r} \vee s) \wedge (q \vee \overline{r} \vee s)$ está en FNC.

© Todos los derechos reservados

2.4. ALGEBRA BOOLEANA

Resulta muy útil describir los circuitos lógico a través de ecuaciones matemáticas, denominadas funciones booleanas, que tienen la particularidad de que las variables y los valores devueltos por la función son booleanos (0,1), para esto se utiliza el Álgebra de Boole.

Sobre las variables booleanas se pueden definir las funciones: suma lógica (+), correspondiente al OR ($\vee$) de la lógica proposicional; producto lógico ($\bullet$) equivalente al AND ($\wedge$); el complemento (') que representa la negación ( ) y la igualdad (=) que representa la relación de equivalencia ($\equiv$). En el caso del producto lógico entre variables diferentes es muy común obviar el operador ($\bullet$), de esta forma (x $\bullet$ y) es lo mismo que xy; en lo sucesivo se utilizará este convenio para evitar el uso excesivo de paréntesis. Así las FND y FNC obtenidas en los ejemplos 2.3.2 y 2.3.3 se expresarían en el álgebra booleana como:

- $q + p' + r'$ *Ejemplo 2.3.2.*
- $(p + r' + s) \bullet (q + r' + s)$ o simplemente $(p + r' + s)\ (q + r' + s)$ *Ejemplo 2.3.3.*

Para cualquier variable booleana, se aplican las siguientes leyes:

1.	**x'' = x**	(Ley de doble negación)
2.	**x + x = x**	(Ley de idempotencia)
3.	**x • x = x**	
4.	**(x + y) + z= x + (y + z)**	(Ley asociativa)
5.	**(xy) z = x (yz)**	
6.	**x + y = y + x**	(Ley conmutativa)
7.	**xy = yx**	
8.	**x + yz = (x + y) (x + z)**	(Ley distributiva)
9.	**x (y + z) = xy + xz**	
10.	**(x + y)' = x'y'**	(Ley de De Morgan)
11.	**(x y)' = x'+ y'**	
12.	**x + 0 = x**	(Ley de identidad)
13.	**x • 1 = x**	
14.	**x + 1 = 1**	(Ley de dominación)
15.	**x • 0 = 0**	
16.	**x • x' = 0**	(Ley inversa)
17.	**x + x'= 1**	
18.	**x + x y = x**	(Ley de absorción)
19.	**x (x+ y) = x**	

Note que las leyes anteriores se corresponden con las estudiadas en la sección 1.6 para la Lógica proposicional.

© Todos los derechos reservados

2.5. FORMAS CANÓNICAS

En los ejemplos 2.3.2 y 2.3.3 se puede observar que en los resultados obtenidos no todos los términos contienen la totalidad de las variables, sin embargo esto no resulta muy efectivo en la simplificación de circuito, a continuación se expondrán algunos conceptos asociados a este planteamiento y cómo solucionarlo.

En el Álgebra Booleana se denomina **término canónico** de una función booleana a las expresiones formadas por un producto (o la suma) en el que todas las variables de la función aparecen una sola vez en forma complementada o sin complementar. Si el término canónico es un producto, se dice que es un **mintérmino** o **producto canónico**, y cuando es una suma, **maxtérmino** o **suma canónica**.

Por tanto, un mintérmino es una expresión lógica de *n* variables que utiliza únicamente el operador de conjunción (AND) y el operador complemento o negación (NOT). Mientras que un maxtérmino contiene solo los operadores de disyunción (OR) y de negación (NOT) para relacionar las *n* variables de la función. Una función de *n* variables tiene a lo sumo 2^n sumas canónicas y 2^n productos canónicos distintos. Para una función f(x,y,z) los términos xy'z, xyz, x'y'z' son ejemplos de mintérminos, mientras que x'+y+z, x'+y+z', x+y+z constituyen maxtérminos de la función.

Todas las funciones lógicas se pueden expresar en forma canónica, como una "suma de mintérminos" o como un "producto de maxtérminos". Esto facilita la simplificación de dichas funciones tan importante en el diseño de circuitos digitales. Cuando una función está expresada como una "suma de mintérminos", también llamada "suma de productos" se dice que está en **Forma canónica disyuntiva (FCD)**. Si la función es un "producto de maxtérminos" o "producto de sumas" entonces está en **Forma canónica conjuntiva (FCC)**.

En ocasiones se desea obtenerse la expresión canónica de una función definida mediante una expresión algebraica. Para ello, se transforma en FNC (o FND) y se multiplica (o suma) cada término no canónico por la suma (o producto) de las variables que faltan y sus inversas, "...• (w + w')" si es una FND y "...+ (w • w')" o simplemente + (w w') si es una FNC, donde w representa la variable ausente. Como ya se explicó el proceso para obtener las formas normales se partirá de haber realizado este paso.

EJEMPLO 2.5.1.

Determinar la Forma canónica de las FN obtenidas en los ejemplos 2.3.2 y 2.3.3.

a) Determinar la Forma canónica disyuntiva de la expresión: q + p' + r'

b) Determinar la Forma canónica conjuntiva de la expresión:
(p + r' + s) (q + r' + s)

© TODOS LOS DERECHOS RESERVADOS

Solución:

a) q(p + p') (r + r') + p' (q + q') (r + r') + r' (p + p') (q + q') = q(pr + pr' + p'r + p'r') + p'(qr + qr' + q'r + q'r') + r'(pq + pq' + p'q + p'q')

= pqr + pqr' + p'qr + p'qr' + p'qr + p'qr' + p'q'r + p'q'r' + pq r' + pq'r' + p'q r' + p'q' r'

Note que existen términos repetidos que están señalados con un mismo color. Al eliminar los términos que se repiten, según la Ley de idempotencia A + A = A, queda la expresión siguiente:

q p r + pqr'+ p'qr + p'qr'+ p'q'r + pq'r'+ p'q' r' (Forma canónica disyuntiva)

b) (p + r' + s + (q q')) (q + r' + s+ (p p')) = (p + r' + s + q) (p + r' + s + q') (p + r' + s + q) (q + r' + s + p')

Note que los dos términos señalados son iguales y según la Ley de idempotencia A • A = A, por tanto la expresión final en Forma canónica conjuntiva es: (p + r' + s + q') (p + r' + s + q) (q + r' + s+ p')

Las formas canónicas también se pueden obtener a partir de la tabla de verdad de una función. En el caso de la forma canónica disyuntiva se tienen en cuenta las combinaciones para las que la función de salida toma valor 1. Estas combinaciones dan lugar a los mintérminos donde las variables con valor 0 se escriben en forma complementada y las de valor 1 sin complementar. Finalmente se suman todos los mintérminos obtenidos.

Ejemplo 2.5.2. Formas canónicas

Exprese en forma de suma de productos la función representada por la tabla de verdad siguiente:

Tabla 2.4: Tabla de verdad, ejemplo 2.5.2.

a	b	c	f (a,b,c)
0	0	0	**1**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

© Todos los derechos reservados

Solución:

La función toma valor 1 en las filas 1, 4 y 7, por tanto la función resultante tendrá solo 3 mintérminos. El término correspondiente a la cuarta fila es a'bc, la variable a está en forma complementada porque tiene valor cero en la combinación, mientras que b y c están en forma directa porque tienen valor 1. Siguiendo este procedimiento y sumando los 3 mintérminos obtenidos la expresión final es:

$$f(a,b,c) = a'b'c' + a'bc + abc'$$

Para obtener la función en forma normal conjuntiva se sigue el procedimiento contrario. Se tienen en cuenta las combinaciones para las que la función de salida toma valor 0. Estas combinaciones dan lugar a los maxtérminos donde las variables con valor 1 se escriben en forma complementada y las de valor 0 sin complementar. Finalmente se multiplican todos los maxtérminos obtenidos.

**Nota*: intente obtener la forma canónica disyuntiva (suma de productos) a partir de la tabla de verdad de la expresión q + p' + r'. Verifique que el resultado sea igual al obtenido en el ejemplo 2.5.1 inciso a.

Ejemplo 2.5.3. Formas canónicas

Exprese como un producto de sumas la función representada por la tabla de verdad del ejemplo 2.5.2.

Solución:

La función toma valor 0 en las filas 2, 3, 5, 6 y 8 para un total de 5 maxtérminos que compondrán la función resultante. El término correspondiente a la fila 8 es (a'+b'+c'), las tres variables se escriben en forma complementada porque tiene valor uno en la combinación, recuerde que se trabaja de forma contraria a la suma de productos. Se utiliza este procedimiento para formar los maxtérminos restantes y luego se multiplican obteniéndose la expresión final siguiente:

$$f(a,b,c) = (a+b+c')\,(a+b'+c)\,(a'+b+c)\,(a'+b+c')\,(a'+b'+c')$$

**Nota*: cuando la salida de la función representada en la tabla de verdad contiene menos 0 que 1 es más conveniente expresar la función como un producto de sumas ya que esta tendría menos términos que si se expresara como suma de productos. En caso de que sean menos los 1 entonces es aconsejable expresar la función en forma de suma de productos.

Las formas canónicas también suelen escribirse en forma numérica basándose en los valores decimales de las combinaciones de la tabla correspondientes a los mintérminos o maxtérminos, según sea el caso. Para una función de *n* variables la forma canónica disyuntiva (suma de productos) se expresa como $\Sigma_n m_i$ siendo m_i el valor decimal de las combinaciones de la tabla de verdad correspondientes a los mintérminos de la función. Para la forma canónica disyuntiva (producto de sumas) se utiliza el símbolo Π, quedando

© Todos los derechos reservados

expresada como $\Pi_n M_i$, donde M_i es el valor decimal de las combinaciones de la tabla de verdad correspondientes a los maxtérminos de la función.

**Nota*: el valor de un número binario se obtiene sumando los valores obtenidos de multiplicar cada dígito por 2^x, donde x representa la posición del dígito numerada de derecha a izquierda, partiendo de 0.
Ejemplo 101= $1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5$

EJEMPLO 2.5.4. FORMAS CANÓNICAS

Escriba en forma numérica las formas canónicas de la función del ejemplo 2.5.2.

Solución:

Primero se debe determinar el valor decimal de cada una de las combinaciones de la tabla de verdad. El valor correspondiente a cada combinación de a, b y c se muestra en la primera columna de la tabla 2.5.

Tabla 2.5: Tabla de verdad con valores decimales incluidos, ejemplo 2.5.4.

valor	a	b	c	f (a,b,c)
0	0	0	0	**1**
1	0	0	1	**0**
2	0	1	0	**0**
3	0	1	1	**1**
4	1	0	0	**0**
5	1	0	1	**0**
6	1	1	0	**1**
7	1	1	1	**0**

Forma canónica disyuntiva:

$f(a, b, c) = \Sigma_3(0,3,6)$

Forma canónica conjuntiva:

$f(a, b, c) = \Pi_3(1,2,4,5,7)$

Note que los términos ausentes en la forma canónica disyuntiva son los utilizados en la forma canónica conjuntiva.

También se puede hacer directamente a partir de las formas canónicas obtenidas en los ejemplos 2.5.2 y 2.5.3 determinando el número decimal correspondiente a cada mintérmino al interpretar las variables presentes como 0 si está complementada o como 1

© TODOS LOS DERECHOS RESERVADOS

si está sin complementar y para los maxtérminos interpretando el valor de las variables como 1 si está complementada y 0 sin complementar.

2.6. SIMPLIFICACIÓN DE CIRCUITOS

Antes de implementar un circuito lógico, es decir, antes de convertir las ecuaciones a compuertas lógicas, se debe reducir al máximo la función booleana que lo describe. Simplificar un circuito significa entonces obtener una expresión con el mínimo posible de términos y variables por términos. La expresión simplificada permite implementar un circuito equivalente al de la expresión inicial pero con menos compuertas y conexiones, por tanto, más óptimo. Se dice que dos circuitos lógicos son equivalentes si y solo si tienen igual cantidad de entradas y de salidas, y por cada una de las entradas se obtienen las mismas salidas, o sea, son equivalentes si realizan la misma función.

En el proceso de simplificación de expresiones en formas canónicas pueden agruparse dos términos eliminando la variable respecto a la que difieren y quedando únicamente la parte común, matemáticamente se expresa como:

- $(a + b + c + ...)\ (a' + b + c + ...) = b + c + ...$
- $a\ b\ c... + a'\ b\ c... = b\ c...$

Estas ecuaciones se deducen de la combinación de la Ley distributiva, la Ley de identidad y la Ley inversa:

- $(a + b + c + ...)\ (a' + b + c + ...) = (a \bullet a') + (b + c + ...) = 0 + (b + c + ...) = b + c + ...$
- $a\ b\ c... + a'\ b\ c... = (a + a')\ (b\ c...) = 1 \bullet (b\ c...) = b\ c...$

EJEMPLO 2.6.1 SIMPLIFICACIÓN DE EXPRESIONES EN FORMA CANÓNICA

Simplifique la expresión siguiente: x' y z' + x' y z + x y z' + x y z

Solución:

El proceso se basa en identificar aquellos términos que difieran en solo una variable manteniendo solo la parte común hasta que no existan elementos que difieran en una única variable.

$$\underbrace{\underbrace{\mathbf{x'}\ \mathbf{y}\ z' + \mathbf{x'}\ \mathbf{y}\ z}_{\mathbf{x'}\ y} + \underbrace{\mathbf{x}\ \mathbf{y}\ z' + \mathbf{x}\ \mathbf{y}\ z}_{\mathbf{x}\ y}}_{y}$$

© TODOS LOS DERECHOS RESERVADOS

Entonces: x' y z' + x' y z + x y z' + x y z = y

Los principales métodos de simplificación de circuitos reiteran este proceso de agrupamiento hasta obtener una expresión que ya no contenga más términos adyacentes. Este material aborda dos métodos de simplificación, el método algebraico y los mapas de Karnaugh, los cuales se explican a continuación.

2.6.1. SIMPLIFICACIÓN ALGEBRAICA

Utiliza los teoremas del Álgebra booleana para la transformación de las expresiones en equivalentes más simples. Aunque es útil tiene algunas desventajas:

- No resulta obvio qué teorema utilizar en cada caso para obtener el mejor resultado.
- El procedimiento es extenso para la simplificación de expresiones de un gran número de términos.
- No garantiza que la expresión obtenida sea irreducible.
- Requiere de mucha experiencia y habilidades en el álgebra.

EJEMPLO 2.6.2. SIMPLIFICACIÓN ALGEBRAICA DE CIRCUITOS

Se desea diseñar un circuito que represente la función x y' z + x y' z' + x' y' z' + x' y' z + x' y z. Realice el proceso de simplificación previo, utilizando el método algebraico.

Solución:

Primero se identifican los términos que difieren en una única variables y se mantienen solo las variables comunes.

x y' z + x y' z' + x' y' z' + x' y' z + x' y z = xy' + **x' y'** z' + **x' y' z** + **x'** y **z**

Note que el término (x' y' z) tiene en común con el término que le antecede las variables (x' y') y difieren solo en la variable "z", mientras que con el término siguiente tiene a (x' z) y difiere en la variable "y". Una forma simple de resolver este caso es aplicando la Ley de idempotencia (x' y' z = x' y' z + x' y' z). Luego se hacen las agrupaciones siguientes:

x y' + x' y' z' + x' y' z + x' y' z + x' y z = x y' + x' y' + x' z

Finalmente:

x y' + x' y' + x' z = **y' + x' z**

© TODOS LOS DERECHOS RESERVADOS

EJEMPLO 2.6.3. SIMPLIFICACIÓN ALGEBRAICA DE CIRCUITOS

Una función de tres variables *f(a,b,c)* toma valor cero cuando la variable *b* vale uno y la variable a no, en los demás casos la función vale cero. Diseñe un circuito simplificado de la función.

Solución:

**Nota*: este ejemplo muestra el proceso completo de diseño de circuitos: especificación del circuito; obtención de la tabla de verdad y la expresión analítica que representa el circuito; simplificación del circuito y por último la implementación de circuito simplificado. A partir de la especificación del circuito expresada en la orden del ejercicio se procede a la construcción de la tabla de verdad.

Paso 1. Realizar la tabla de verdad de la función.

Tabla 2.6: Tabla de verdad del ejemplo 2.6.3.

a	b	c	f (a,b,c)
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Paso 2. Expresar en forma canónica.

Como los valores de salida de la función presentan menos 0 que 1 se debe expresar la función en forma conjuntiva: (a + b' + c) (a + b' + c').

Paso 3. Minimizar la expresión obtenida en forma canónica.

(<u>a + b'</u> + c) (<u>a + b'</u> + c')

= a + b'

© TODOS LOS DERECHOS RESERVADOS

Paso 4. Representar el circuito.

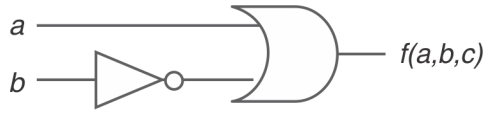


Figura 2.7. Circuito simplificado del ejemplo 2.6.3.

2.6.2. Mapas de Karnaugh

Es un método gráfico de representar la tabla de verdad que utiliza el reconocimiento de patrones gráficos para la simplificación de expresiones booleanas. El uso de los mapas de Karnaugh es un método más simple y ordenado que el algebraico, con pasos bien definidos, además brinda mayor seguridad del grado de simplificación de la expresión obtenida. Sin embargo este método no es viable en diseños complejos, ya que a partir de 6 variables los mapas se hacen demasiado complicados y pierden utilidad. Para circuitos de más de 6 variables se utiliza el método Quine – McCluskey que no es objeto de estudio de este material ya que comúnmente se emplea de forma computarizada. Los mapas de Karnaugh se construyen utilizando la tabla de verdad de la expresión booleana o a partir de una expresión directamente. Los pasos para la simplificación utilizando mapas de Karnaugh son los siguientes:

Paso 1:

En función de la cantidad de entradas o variables se elabora una tabla de 2^n cuadrículas, donde n constituye el número de variables de la función. Las cabeceras de las columnas y las filas corresponden a las combinaciones binarias de los valores de la(s) variable(s) indicadas en el extremo superior izquierdo. Las combinaciones binarias se organizan de forma que entre una columna y la contigua cambie el valor de una sola variable y lo mismo para las filas. De esta forma se facilita la simplificación ya que garantiza que las celdas contiguas sean consideradas adyacentes, esto es clave en el proceso de simplificación que inicia en el paso 3. La figura 2.8 muestra el sistema de cuadrículas resultante para una función de 2, 3, 4 y 5 variables respectivamente.

© Todos los derechos reservados

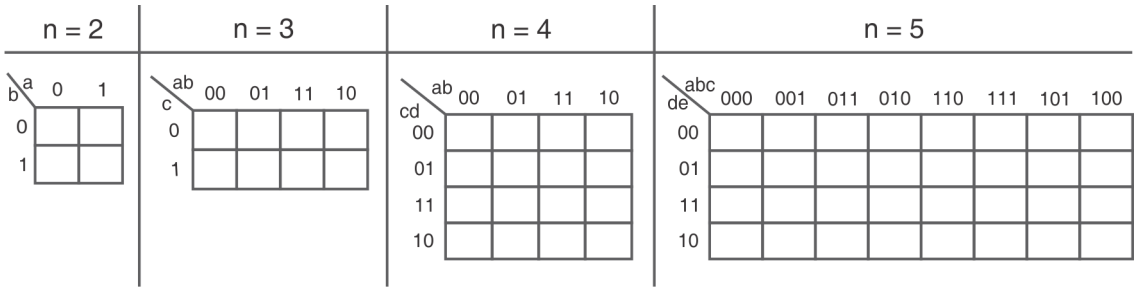


Figura 2.8. Esquema de los mapas de Karnaugh según el número de variables.

Se entiende por celdas adyacentes aquellas que difieren en el valor de una sola de las variables. Por ejemplo en el sistema de cuadrículas correspondiente a 4 variables de la figura 2.8, para la casilla inferior derecha a=1, b=0, c=1 y d=0. Para la casilla que está a su izquierda los valores de las variables son a=1, b=1, c=1 y d=0. Al comparar los valores de las variables correspondientes a ambas casillas se puede apreciar que solo cambia una de las tres variables, la b. Lo mismo ocurre si se realiza un desplazamiento horizontal o verticalmente hacia cualquier otra casilla contigua.

Paso 2:

Luego de confeccionar el sistema de cuadrículas se procede a rellenarlas. Si se parte de la tabla de verdad a cada casilla se le asigna el valor de la salida correspondiente a los valores de las entradas determinado por las cabeceras de la columna y la fila a la que pertenece. Por ejemplo para la primera fila de la tabla de verdad de la figura 2.9, a, b y c tienen valor cero por lo que el valor de la salida, que es 1, se ubica en la casilla superior izquierda cuya cabeceras coinciden con estos valores. En la segunda fila de la tabla de verdad a=0, b=0 y c=1 por lo que la salida (uno) se sitúa en la segunda casilla de izquierda a derecha de la primera fila como se puede observar en la figura 2.9.

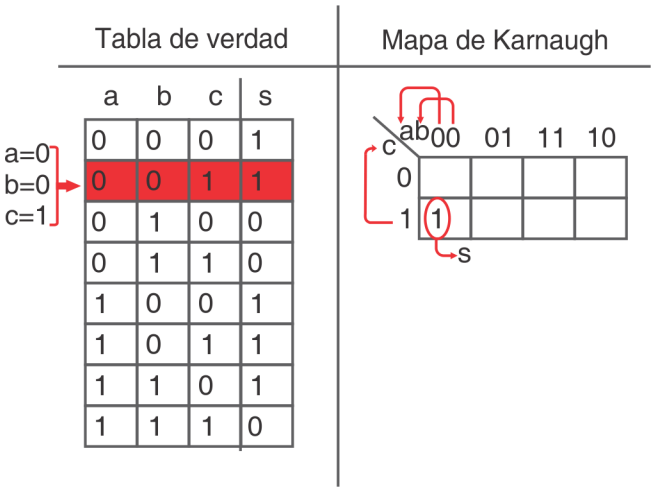


Figura 2.9. Confección del mapa de Karnaugh usando la tabla de verdad.

© Todos los derechos reservados

Se sigue el mismo procedimiento para todas las filas de la tabla de verdad hasta completar el mapa de Karnaugh como muestra la figura 2.10.

Tabla de verdad

a	b	c	s
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Mapa de Karnaugh

c \ ab	00	01	11	10
0	1	0	0	1
1	1	0	1	1

Figura 2.10. Mapa de Karnaugh terminado.

© Todos los derechos reservados

Si en vez de la tabla de verdad se tiene la expresión booleana en forma de suma de productos entonces se rellenan con 1 las casillas correspondientes a los términos que aparecen en la expresión y el resto con 0. Para esto se utiliza la expresión en su forma canónica, donde las variables negadas se consideran con valor 0 y las que no aparecen negadas con valor 1. La cantidad de 1 existentes en el mapa de Karnaugh es equivalente a la cantidad de términos sumados en la función booleana. La figura 2.11 muestra un ejemplo con el resultado de este proceso, note que la función representada tiene el mismo mapa de Karnaugh de la figura 2.10 y por tanto le corresponde la misma tabla de verdad.

Función booleana

f(a,b,c)=a'b'c' + a'b'c + ab'c' + ab'c + abc'

<u>Forma numérica:</u>
f(a,b,c)= $\Sigma_3(0,1,4,5,6)= \Pi_3(2,3,7)$

Mapa de Karnaugh

c \ ab	00	01	11	10
0	1	0	1	1
1	1	0	0	1

Figura 2.11. Confección del mapa de Karnaugh usando una función booleana.

Paso 3:

Con el mapa de Karnaugh completado se comienza el proceso de simplificación, para esto se señalan los agrupamientos de 1 con un tamaño potencia de dos, es decir, grupos de un 1, dos 1, cuatro 1 y así sucesivamente. Los agrupamiento deben ser de 1 contiguos horizontal o

verticalmente (celdas adyacentes), no pueden aparecer 0 dentro de los agrupamientos. Si se enrolla el mapa, horizontal o vertical, en forma de cilindro de manera que se topen los extremos, se puede apreciar que la primera y la última fila de casillas se consideran adyacentes, así como la primera con la última columna de casillas, La figura 2.12 muestra algunos ejemplos de agrupaciones.

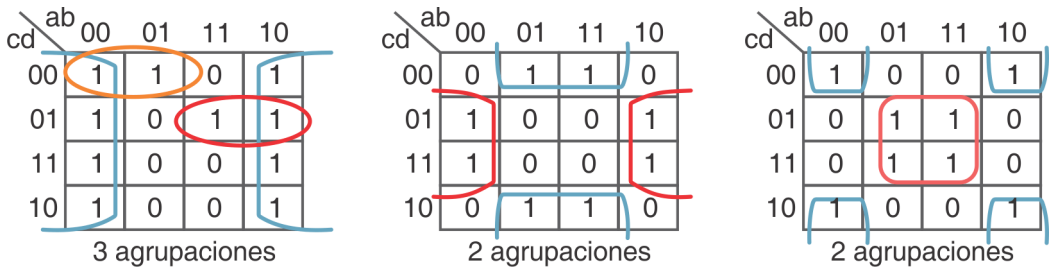
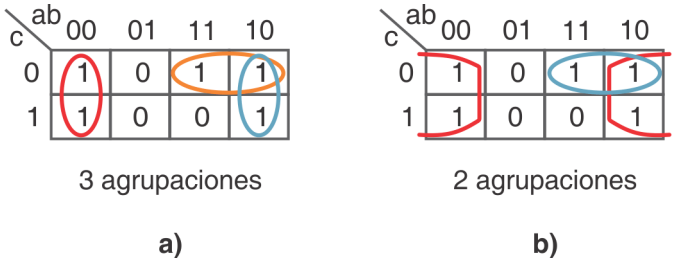


Figura 2.12. Ejemplos de agrupaciones.

Existen diferentes simplificaciones posibles para un mismo mapa de Karnaugh en dependencia de los agrupamientos que se seleccionen, por lo que es necesario escoger los agrupamientos de una forma óptima para lograr la expresión más simplificada. Cada agrupación identificada corresponde a un término de la expresión simplificada. Por tanto para obtener el máximo de simplificación se debe seleccionar el menor número de agrupaciones, para esto las agrupaciones formadas deben tener el mayor número de 1 posible, sin importar que se solapen entre ellas. La figura 2.13 muestra 2 posibles formas de agrupamientos para el mapa de Karnaugh obtenido en la figura 2.10 y 2.11.



a) **b)**

Figura 2.13. a) Se seleccionan 3 agrupamientos por lo que la expresión simplificada contiene 3 términos. b) Se identifican 2 agrupamientos, este es más óptimo porque contiene menos agrupaciones y estas a su vez contienen más 1 que las agrupaciones de la figura 2.13a.

Paso 4:

Para obtener la expresión simplificada por cada agrupación se eliminan las variables que cambien de valor, apareciendo complementadas y sin complementar, puesto que su valor no afecta el resultado. Con este paso se aplica de una forma gráfica intuitiva las ecuaciones de simplificación vistas al principio de la sección 2.5. En agrupamientos de 2^m elementos se eliminan *m* variables,

© Todos los derechos reservados

es decir, los términos de la expresión simplificada están compuestos por ($n - m$) literales, siendo n el número de variables de entrada del mapa de Karnaugh. Para la figura 2.13b el término correspondiente al agrupamiento de tamaño 4 (2^2: m=2) contiene un solo literal puesto que n=3 (son 3 entradas: a, b, c) y por tanto 3 – 2= 1.

La expresión simplificada se forma sumando los términos correspondientes a cada agrupamiento, donde un término constituye la multiplicación de las variables comunes (las que no se eliminan) del agrupamiento respectivo. Para el mapa de Karnaugh de la figura 2.13b note que en la agrupación marcada en rojo la única variable que no cambia su valor es *b*, que está en su forma complementada por lo que en la expresión iría *b*' y en la agrupación de color azul las variables comunes en valor son a sin complementar y *c* complementada. Finalmente se suman ambos términos y la expresión simplificada sería $f(a,b,c) = b' + ac'$.

Una vez obtenida la expresión simplificada se puede elaborar un circuito capaz de cumplir con la función original de forma más óptima. El circuito correspondiente a la expresión $f(a,b,c) = b' + ac'$ es el que se muestra en la figura 2.14.

Figura 2.14. Circuito simplificado de la función f(a,b,c)= $\Sigma_3(0,3,4,6,7)$.

© Todos los derechos reservados

Una misma función puede escribirse como suma de productos o como producto de sumas, en ambos casos se obtiene el mismo mapa de Karnaugh. La explicación de los pasos anteriores se realizó sobre la base de la simplificación de una expresión en forma de "suma de productos" pero si se parte de una expresión booleana en forma de producto de sumas en el paso 2 se realiza el proceso contrario. En el mapa de Karnaugh se rellenan con 0 las casillas correspondientes a los términos que aparecen en la expresión y el resto con 1. Se debe utilizar la expresión en su forma canónica, las variables negadas se consideran con valor 1 y las que no aparecen negadas con valor 0. La cantidad de 0 existentes en el mapa de Karnaugh es equivalente a la cantidad de términos multiplicados en la función booleana.

En el paso 3 se agrupan los 0 en vez de los 1, siguiendo el mismo criterio explicado. Luego la expresión final sería el producto de los términos compuestos por la suma de las variables comunes del agrupamiento. Se debe recordar que a diferencia de la "suma de productos", en el "producto de sumas" las variables con valor 1 se toman como el complemento y las de valor 0 como no complementadas. El ejemplo 2.6.4 muestra el proceso de simplificación de un producto de sumas.

La óptima simplificación de la expresión depende de la cantidad de agrupaciones, lo que influye en la cantidad de términos de la expresión y de la cantidad de elementos que abarquen las agrupaciones, que determina el número de literales dentro del término. Por ejemplo, para

simplificar como “producto de sumas” la expresión representada por el mapa de Karnaugh de la figura 2.13 el resultado de los pasos 3 y 4 sería el mostrado en la figura 2.15a con el correspondiente circuito de la figura 2.15b:

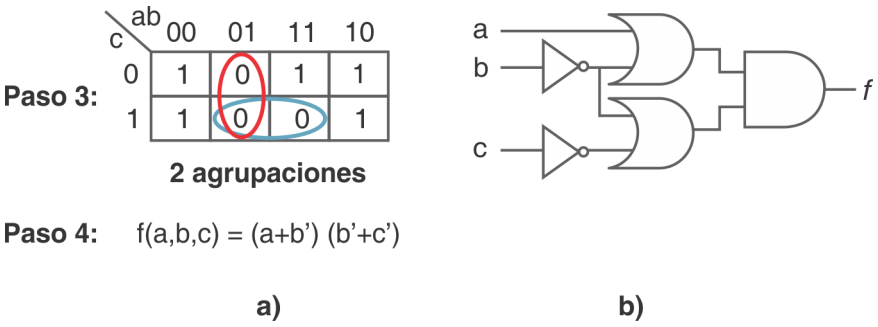


Figura 2.15. Circuito simplificado como producto de sumas.

Note que a pesar de haber menos 0 que 1 en el mapa de Karnaugh el circuito del producto de sumas tiene mayor número de conexiones y compuertas. Esto se debe a que, aunque en los mapas de las figuras 2.13b y 2.15a existe el mismo número de agrupaciones, en la primera aparece una agrupación con mayor número de elementos que las agrupaciones de la figura 2.15a. Por tanto en este caso es más óptimo el circuito de la suma de productos.

Ejemplo 2.6.4. Simplificación de circuitos

Simplifique y represente el circuito correspondiente a la función booleana siguiente:

f(a,b,c,d)= (a+b'+c+d) (a'+b'+c+d) (a+b'+c'+d') (a'+b'+c'+d') (a+b'+c'+d) (a'+b'+c'+d)

Solución:

El primer paso sería elaborar el mapa de Karnaugh (figura 2.16), teniendo en cuenta que al estar la función en forma de producto de suma los términos corresponden a 0 en el mapa y los valores de las entradas están negados.

cd \ ab	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	1	0	0	1
10	1	0	0	1

Figura 2.16. Mapa de Karnaugh del ejemplo 2.6.4.

© Todos los derechos reservados

Con el mapa de Karnaugh se hacen las agrupaciones de 0 como se muestra en la figura 2.17.

cd \ ab	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	1	0	0	1
10	1	0	0	1

2 agrupaciones

Figura 2.17. Agrupaciones del mapa de Karnaugh el ejemplo 2.6.4.

A partir de las agrupaciones identificadas se obtiene una expresión simplificada de 2 términos. El correspondiente a la agrupación de 2 ceros (tamaño 2^1: m=1) tiene 3 literales según la fórmula (n – m) donde n vale 4 (variables de entrada). El término de la agrupación de 4 ceros (tamaño 2^2: m=2) consta de 2 literales. La expresión simplificada es (b'+c+d) (b'+c').

En la figura 2.18 Se muestra el circuito correspondiente a la función obtenida.

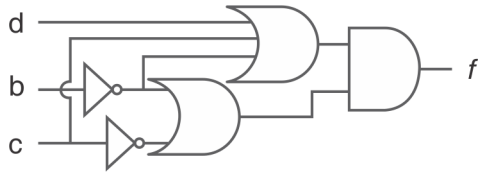


Figura 2.18. Circuito de la función (b'+c+d) (b'+c').

Ejemplo 2.6.5. Simplificación de circuitos

Diseñe un circuito, con la menor cantidad de compuertas y conexiones posibles, que cumpla con la función *f* (a,b,c) representada por tabla de verdad siguiente:

© Todos los derechos reservados

Tabla 2.7: Tabla de verdad del ejemplo 2.6.5.

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Solución:

Primero se debe obtener la expresión booleana representada por la tabla de verdad, luego simplificarla, para lo cual se utilizará un mapa de Karnaugh y finalmente se elabora el circuito con la expresión simplificada obtenida.

Paso 1: obtener la función booleana no simplificada a partir de la tabla de verdad.

Suma de productos

$f(a,b,c) = ab'c' + ab'c + abc = \Sigma_3(4,5,7)$

Producto de sumas

$f(a,b,c) = (a+b+c)\ (a+b+c')\ (a+b'+c)\ (a+b'+c')\ (a'+b'+c) = \Pi_3(0,1,2,3,6)$

Paso 2: elaborar mapa de Karnaugh.

c \ ab	00	01	11	10
0	0	0	0	1
1	0	0	1	1

Figura 2.19. Mapa de Karnaugh del ejemplo 2.6.5.

Paso 3: identificar las agrupaciones.

Según el mapa de Karnaugh obtenido es evidente que resulta más óptimo simplificar la expresión como producto de sumas ya que sus agrupaciones reúnen más elementos. Para visualizar esto véanse las posibles agrupaciones para ambos casos en la figura 2.20. Las

© Todos los derechos reservados

agrupaciones para la suma de productos aparecen señaladas en azul y en rojo las agrupaciones para el producto de sumas.

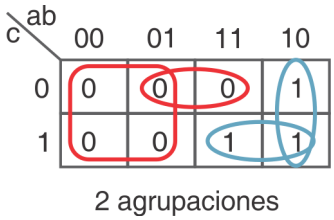


Figura 2.20. Agrupaciones para el producto de sumas (en rojo) y para la suma de productos (en azul).

Paso 4: Obtener expresión simplificada.

La función simplificada como producto de sumas es f(a,b,c)= a(b'+ c), mientras que la simplificada como suma de productos es f(a,b,c)= a'b + a'c' que implica una compuerta más y un mayor número de conexiones lo cual confirma lo expresado en el paso 3. Por tanto el circuito que cumple con los requerimientos del ejercicio es el correspondiente al producto de sumas (figura 2.21).

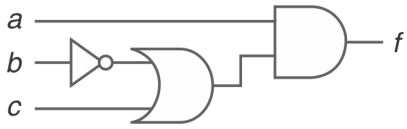


Figura 2.21. Circuito de la función f(a,b,c)= a(b'+ c).

© Todos los derechos reservados

Resumen

© Todos los derechos reservados

Lógica de predicados 3

Objetivos

- Analizar los límites de la Lógica proposicional.
- Definir los elementos básicos de la Lógica de predicado, con hincapié en los operadores de cuantificación.
- Interpretar fórmulas de la Lógica de predicados para la determinación de su valor veritativo.
- Analizar el concepto de equivalencia en la Lógica de predicados.
- Plantear las leyes de equivalencia para los cuantificadores.

© Todos los derechos reservados

3.1. INTRODUCCIÓN

Vimos en el capítulo 1 que mediante la Lógica Proposicional podemos modelar gran cantidad de enunciados a los cuales podemos atribuir un valor de verdad, es decir, proposiciones.

El hecho de que una proposición debe tener uno y solo un valor de verdad queda determinado a partir de la propia definición del enunciado.

Cada uno de los siguientes ejemplos son proposiciones:

1. $(-2)^2 + 4(-2) + 4 = 0$
2. Los estudiantes Pedro, Juan y María aprobaron el examen.
3. Existen estudiantes que aprobaron el examen.
4. Todos los estudiantes aprobaron el examen.

Podemos asegurar que en el ejemplo 1 todos los enunciados son proposiciones pues no cabe duda de que podemos asignarle un valor de verdad a cada uno. Un simple cálculo puede establecer que la proposición 1 es verdadera. Pudiera ser cierto o no que tanto Juan, Pedro como María aprobaron el examen, como dice la proposición 2; en caso de ser verdadera también lo será la proposición 3, que también pudiera ser verdadera aun cuando 2 no lo fuera, pues no necesariamente Juan, Pedro y María son los únicos que se examinaron, pero en tal caso, cuando Juan o Pedro o María no aprobaran el examen, sería falsa entonces la proposición 4.

Como se puede notar en los enunciados analizados, para las proposiciones 1 y 2 podemos establecer su veracidad sin más información que el cálculo de la expresión para el caso de la primera, o saber si realmente Juan, Pedro y María aprobaron el examen, para el segundo caso.

Algo distinto ocurre con las proposiciones 3 y 4 del ejemplo 1. En la primera, para asegurar que es falsa, se debe verificar que todos los estudiantes que se examinaron no lograron aprobar, y solo en tal caso podemos entonces asegurar que no existen estudiantes que aprobaron el test (la veracidad de la negación); si en nuestro proceso de verificación encontramos un estudiante que aprobó el test entonces paramos y decimos que es verdadera.

Lo mismo tendremos que hacer si queremos establecer que es cierta la proposición 4.

De modo que existen enunciados cuya veracidad depende de la naturaleza de los objetos o constantes que involucra, tomados estos de un conjunto bien determinado. Analicemos los siguientes enunciados.

5. $x^2 + 4x + 4 = 0$
6. Aprobó el test

Está claro que no podemos decir si el enunciado 5 es cierto o no, pues esto depende el valor que pueda tomar la variable x. Para $x = -2$ el enunciado se transformará en la proposición 1 y sería en tal caso cierta; si asignamos a x un valor distinto de -2 sería evidentemente falsa. O podríamos reformular el enunciado como sigue:

Existe x tal que $(-2)^2 + 4(-2) + 4 = 0$ y como sabemos que la ecuación tiene una solución real para $x = -2$ podemos decir que es cierta, claro está, si es que analizamos la existencia de x dentro del conjunto de los números reales, pues otra cosa ocurriría si restringimos el domino de la variable x, digamos a los reales positivos R^+, de modo que no existe un número positivo que satisfaga la igualdad, entonces sería falsa. Un análisis similar se puede realizar si reformulamos de la siguiente manera:

Para todo x se cumple que $x^2 + 4x + 4 = 0$

En igual situación nos encontramos frente al enunciado 6. No podemos decir si es cierto o no, a no ser que se especifique quién aprobó, ¿acaso Juan?, ¿María? Es más, el enunciado es tan escueto que cabría la duda ¿se refiere a los estudiantes?; y aun cuando se refiera a los estudiantes volverían las primeras interrogantes, o estas, ¿se refiere a todos? ¿o si alguno aprobó?.

En los ejemplos anteriores vimos enunciados para los cuales no podemos establecer un valor de verdad, pues involucran variables u objetos indeterminados (lo que responde a la pregunta quién en cuanto al enunciado 6). Para poder establecer la veracidad de estos enunciados debemos establecer un dominio de valores posibles a tomar (digamos, el conjunto de estudiantes) y solo si asignamos uno de estos valores o preguntamos por alguno, o por todos, podremos decir si el enunciado es cierto o no.

La Lógica proposicional resulta insuficiente para expresar gran parte de las definiciones o proposiciones de las matemáticas, las ciencias de la computación y del lenguaje natural. Existen

© Todos los derechos reservados

enunciados cuya estructura interna es más compleja de lo que el lenguaje de la lógica proposicional nos permite analizar a partir de su traducción desde el lenguaje natural.

Gottlob Frege (1848-1925) matemático y filósofo alemán. Enfrentándose a la insuficiencia de los sistemas lógicos disponibles, inventó muchas notaciones simbólicas, como cuantificadores y variables, estableciendo así las bases de la lógica matemática moderna, a través del desarrollo de la llamada Lógica de Predicados.

La lógica de predicados también llamada lógica de primer orden o cálculo de predicados, es un sistema formal diseñado para estudiar la inferencia en los lenguajes de primer orden. Los lenguajes de primer orden son, a su vez, lenguajes formales con cuantificadores. La lógica de primer orden tiene el poder expresivo suficiente para definir a prácticamente todas las matemáticas.

Para entender el lenguaje de la Lógica de predicados primeramente se introduce el concepto de predicado y los cuantificadores. Los cuantificadores tienen una gran utilidad ya que permiten indicar si todos o al menos un elemento de un conjunto dado cumple con una determinada propiedad. Luego de introducir la noción de predicado y los cuantificadores, se define un lenguaje formal para la Lógica de predicados y un método para evaluar el valor de verdad de una expresión de dicho lenguaje.

© Todos los derechos reservados

3.2. PREDICADOS

A menudo, en matemáticas y programas de ordenador se encuentran enunciados en los que se incluyen variables, como:

$x > 3$, $x = y + 3$ y $x + y = z$.

Estos enunciados no son ni verdaderos ni falsos si no se especifican los valores de las variables. En esta sección discutiremos las maneras en que las proposiciones pueden producir tales enunciados.

El enunciado "*x* es mayor que 3", tiene dos partes. La primera parte, la variable *x*, es el sujeto del enunciado. La segunda parte –el predicado, "es mayor que 3" hace referencia a una propiedad que puede tener el sujeto. Podemos denotar el enunciado "*x* es mayor que 3" por *P(x)*, donde *P* denota el predicado «es mayor que 3» y *x* es la variable. La sentencia *P(x)* se dice también que es el valor de la ***función proposicional*** *P* en *x*. Una vez que se le haya asignado un valor a la variable *x*, la sentencia *P(x)* se convierte en una proposición y tiene un valor de verdad. Considera el siguiente ejemplo.

Ejemplo 3.2.1.

Si $P(x)$ denota el enunciado "$x > 3$". ¿Cuáles son los valores de verdad de $P(4)$ y $P(2)$?

Solución: obtenemos la sentencia $P(4)$ haciendo $x = 4$ en el enunciado "$x > 3$". Por tanto, $P(4)$, que es el enunciado "$4 > 3$", es verdadero. Sin embargo, $P(2)$, el enunciado «$2 > 3$», es falso.

Podemos también tener sentencias que incluyan más de una variable. Por ejemplo, considera el enunciado $x = y + 3$. Podemos denotar esta sentencia por $Q(x, y)$, donde x e y son variables y Q es el predicado. Cuando se asignan valores a x e y, la sentencia $Q(x, y)$ tiene una tabla de verdad.

Ejemplo 3.2.2.

$Q(x, y)$ denota el enunciado $x = y + 3$ ¿Cuáles son los valores de verdad de las proposiciones $Q(1, 2)$ y $Q(3, 0)$?

Solución: para obtener $Q(1, 2)$ hacemos $x = 1$ e $y = 2$ en la sentencia $Q(x. y)$. Por tanto, $Q(1, 2)$ es el enunciado «$1 = 2 + 3$», que es falso. La sentencia $Q(3, 0)$ es el enunciado «$3 = 0 + 3$», que es verdadera.

Ejemplo 3.2.3.

¿Cuáles son los valores de verdad de las proposiciones $R(1, 2, 3)$ y $R(0,0,1)$? Siendo $R(x, y, z)$: $x + y = z$

Solución: la proposición $R(1, 2, 3)$ se obtiene haciendo $x = 1$, $y = 2$ y $z = 3$ en la sentencia $R(x, y, z)$. Vemos que $R(1, 2, 3)$ es el enunciado «$1 + 2 = 3$», que es verdadero. También se ve que $R(0,0,1)$, el enunciado "$0 + 0 = 1$", es falso.

Como ya se comentaba los predicados establecen propiedades de individuos (como la propiedad de *ser un número primo*) y relaciones entre estos (como la relación de *estudiar en*). De esta forma la Lógica de predicados permite analizar aquellas expresiones, utilizadas habitualmente en las matemáticas, cuya validez no puede ser definida dentro del ámbito de la Lógica proposicional.

En una forma más cercana a la notación matemática de una función, se puede indicar que un objeto x satisface cierta propiedad P con la notación $P(x)$. De este modo la expresión "x es un número primo" puede representarse como $P(x)$.

© Todos los derechos reservados

Definición 3.2.1: una sentencia de la forma $P(x_1, x_1,..., x_n)$ es el valor de la ***función proposicional P*** en la n-tupla $(x_1, x_1,...,x_n)$ *P* se llama también ***predicado***.

Como muestra el siguiente ejemplo, las funciones proposicionales se usan en programas de ordenador.

EJEMPLO 3.2.4.

Considera la sentencia.

if ($x > 1$)

$x = x + 3$

Cuando el programa llega a esta línea, el valor de la variable *x* en este punto de la ejecución se inserta en *P(x)*, que es $x > 1$. Si *P(x)* es verdadera para este valor de *x*, la sentencia de asignación $x = x + 3$ se ejecuta, por lo que el valor de x se incrementa en 3. Si *P(x)* es falsa para este valor de *x*, la sentencia de asignación no se ejecuta, por lo que el valor de *x* no cambia.

Formalizando y generalizando la definición de predicado tendríamos que:

Definición 3.2.2: sea ***D*** un conjunto dado y ***S*** un conjunto de valores atribuibles, un ***predicado o función proposicional n-ario*** es una función del conjunto D^n en el conjunto de valores atribuibles ***S***.

$P: D^n \rightarrow S$

El conjunto de valores posibles de las variables que transforman una determinada expresión en una proposición recibe el nombre de ***dominio de discurso (D)***. Por ejemplo en el inciso a del ejemplo 3.2.4 el predicado *P(x)* tiene como dominio de discurso *D*, el conjunto de los números naturales mayores que 1, ya que para cada *x* en *D* el enunciado *P(x)* adquiere un valor de verdad, donde *P*(3) es verdadera y *P*(1) es falsa, por ejemplo.

Como el lenguaje natural no es apropiado para la formalidad científica, no lo usaremos como en el ejemplo para definir nuestros enunciados. En lo adelante, si queremos denotar que un objeto x satisface cierta propiedad *R*, bastaría solo con denotar (*x*).

3.3. CUANTIFICADORES

Cuando a todas las variables de una función proposicional se le han asignado valores, la sentencia resultante se convierte en una proposición con un cierto valor de verdad. No obstante, hay otra forma importante, llamada ***cuantificación***, de crear una proposición a partir de una función proposicional. Se discutirán en este apartado dos tipos de cuantificación, a saber,

© TODOS LOS DERECHOS RESERVADOS

cuantificación universal y cuantificación existencial. El área de la lógica que trata con predicados y cuantificadores se llama cálculo de predicados o lógica de primer orden.

EJEMPLO 3.3.1.

Analizar las siguientes proposiciones:

a) *q*: Existen personas con problemas cardíacos menores de 30 años.

b) *r*: Todos los trabajadores asistieron a la reunión.

Para asegurar que *q* es falsa, se debe verificar que todas las personas menores de 30 años no presentan problemas cardíacos, solo en tal caso se podrá entonces asegurar que no existen personas con problemas cardíacos menores de 30 años; si en el proceso de verificación se encontrara al menos una persona con problemas cardíacos sería suficiente para decir que *q* es verdadera. Lo mismo ocurre con el enunciado del inciso *c*, si se quiere establecer que es cierto.

Del ejemplo 3.3.1 se deduce que para otorgar un valor de verdad a un predicado basta con prefijar, para cada una de las variables contenidas en el predicado, una de las dos expresiones siguientes (o sus equivalentes):

"Para toda *variable*," (cuantificador universal).

"Existe *variable* tal que" (cuantificador existencial).

© TODOS LOS DERECHOS RESERVADOS

3.3.1. CUANTIFICADOR UNIVERSAL

Muchas sentencias matemáticas imponen que una propiedad es verdadera para todos los valores de una variable en un dominio particular, llamado el universo de discurso o dominio. Tales sentencias se expresan utilizando un cuantificador universal. La cuantificación universal de una función proposicional es la proposición que afirma que *P(x)* es verdadera para todos los valores de *x* en el dominio. El dominio especifica los posibles valores de la variable *x*.

Definición 3.3.1: la cuantificación universal de *P(x)* es la proposición: "*P(x)* es verdadera para todos los valores del dominio de discurso".

La notación: $\forall(x)P(x)$ denota la cuantificación universal de $P(x)$. Aquí llamaremos al símbolo $\forall$ **el cuantificador universal**. La proposición $\forall(x)P(x)$ se lee como: para todo x $P(x)$, para cada x $P(x)$ o para cualquiera x $P(x)$.

EJEMPLO 3.3.2.

Si quisiéramos llevar al lenguaje de la lógica de predicado la siguiente expresión: *Todos los hombres son mortales*.

Solución: tendríamos que declarar dos predicados de la siguiente forma:

H(x): x es un hombre.

M(x): x es mortal.

La expresión resultaría: $\forall(x)\lceil H(x) \rightarrow M(x)\rceil$

EJEMPLO 3.3.3.

Sea *Q(x)* el enunciado $x < 2$. ¿Cuál es el valor de verdad de la cuantificación $\forall(x)\ Q(x)$, donde el dominio consiste en todos los números reales?

Solución: *Q(x)* no es verdadera para todos los números reales *x*. Por ejemplo, *Q(3)* es falsa. Por tanto, $\forall(x)\ Q(x)$ es falsa.

Cuando todos los elementos del dominio se pueden enumerar -escribiéndolos, por ejemplo, como $x_1, x_2, \ldots, x_n$ se sigue que la cuantificación universal.

$\forall(x)\ P(x) \equiv P(x_1) \wedge P(x_2) \wedge \ldots \wedge P(x_n)$

EJEMPLO 3.3.4.

¿Cuál es el valor de verdad de $\forall(x)\ P(x)$, donde *P(x)* es el enunciado $x^2 < 10$ y el dominio consiste en los enteros positivos menores o iguales que 4?

Solución: la sentencia $\forall(x)\ P(x)$ es lo mismo que la conjunción:

$P(1) \wedge P(2) \wedge P(3) \wedge P(4)$

puesto que el dominio consiste en los enteros 1, 2, 3 y 4. Como *P(4)*, la sentencia $4^2 < 10$, es falsa, se sigue que la sentencia $\forall(x)\ P(x)$ es falsa.

Para mostrar que una sentencia de la forma $\forall(x)\ P(x)$ es falsa, donde *P(x)* es una función proposicional, sólo necesitamos encontrar un valor de *x* del dominio para el cual *P(x)* sea falsa. Este valor de *x* se llama contraejemplo de la sentencia $\forall(x)\ P(x)$. Buscar contraejemplos de

© TODOS LOS DERECHOS RESERVADOS

sentencias que contienen al cuantificador universal es una actividad importante en el estudio de las matemáticas, como veremos en las secciones siguientes.

3.3.2. CUANTIFICADOR EXISTENCIAL

Muchas sentencias matemáticas afirman que hay un elemento con una cierta propiedad. Tales sentencias se expresan mediante cuantificadores existenciales. Con un cuantificador existencial formamos una proposición que es verdadera si y sólo si $P(x)$ es verdadera para al menos un valor de x en el dominio.

Definición 3.3.2: la cuantificación existencial de $\boldsymbol{P(x)}$ es la proposición: "Existe un elemento x en el dominio tal que $\boldsymbol{P(x)}$ es verdadera".

Usamos la notación $\exists(x)P(x)$ para la cuantificación existencial de $P(x)$. El símbolo $\exists$ se denomina cuantificador existencial. La cuantificación existencial se lee como:

"Hay un x tal que $P(x)$"

"Hay al menos un x tal que $P(x)$"

o "Para algún x $P(x)$".

EJEMPLO 3.3.5.

Sea $P(x)$ el enunciado $x > 3$. ¿Cuál es el valor de verdad de la cuantificación $\exists(x)P(x)$, donde el dominio consiste en todos los números reales?

Solución: como $x > 3$ es verdadero, por ejemplo, para $x = 4$, la cuantificación existencial de $P(x)$, es decir, $\exists(x)P(x)$, es verdadera.

EJEMPLO 3.3.6.

Sea $Q(x)$ el enunciado $x = x + 1$. ¿Cuál es el valor de verdad de la cuantificación $\exists(x)P(x)$, donde el dominio consiste en todos los números reales?

Solución: como $Q(x)$ es falsa para todo número real x, la cuantificación existencial de $Q(x)$, que es $\exists(x)P(x)$, es falsa.

Cuando todos los elementos del dominio se pueden enumerar escribiéndolos, por ejemplo, como $x_1, x_2,..., x_n$ se sigue que la cuantificación existencial.

© TODOS LOS DERECHOS RESERVADOS

$\exists(x)P(x) \equiv P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n)$

puesto que esta disyunción es verdadera si, y sólo si, al menos uno de $P(x_1)$, $P(x_2)$,..., $P(x_n)$ es verdadera.

EJEMPLO 3.3.7.

¿Cuál es el valor de verdad de $\exists(x)P(x)$, donde $P(x)$ es el enunciado $x^2 > 10$ y el dominio consiste en los enteros positivos menores o iguales que 4?

Solución: como el dominio es {1, 2, 3, 4}, la proposición $\exists(x)P(x)$ es lo mismo que la disyunción.

$P(1) \vee P(2) \vee P(3) \vee P(4)$.

Como $P(4)$, que es el enunciado o $4^2 > 10$ es verdadera, se sigue que $\exists(x)P(x)$ es verdadera.

En la tabla 3.1 se resume el significado de los cuantificadores universales y existenciales.

Tabla 3.1: Cuantificadores.

Sentencia	¿Cuándo es verdadera?	¿Cuándo es falsa?
$\forall(x)P(x)$	$P(x)$ es verdadera para toda.	Hay un x para el que $P(x)$ es falsa.
$\exists(x)P(x)$	Hay un x para el que $P(x)$ es verdadera.	$P(x)$ es falsa para todo x.

Cuando se quiere determinar el valor de verdad de una cuantificación, a veces es útil realizar una búsqueda sobre todos los posibles valores del dominio. Supongamos que hay n objetos en el dominio de la variable x. Para determinar si $\forall(x)\ P(x)$ es verdadera, podemos barrer los n valores de x y ver si $P(x)$ es verdadera para todos ellos. Si encontramos un valor de x para el cual $P(x)$ es falsa, habremos demostrado que $\forall(x)\ P(x)$ es falsa. En caso contrario, $\forall(x)\ P(x)$ es verdadera. Para ver si $\exists(x)P(x)$ es verdadera, barremos la n posible de x buscando algún valor para el cual $P(x)$ sea verdadera. Si encontramos uno, entonces $\exists(x)P(x)$ es verdadera. Si no encontramos tal valor de x, habremos determinado que $\exists(x)P(x)$ es falsa. (Observa que este procedimiento de búsqueda no puede ser aplicado si el dominio se compone de infinitos elementos. Aun así, sigue siendo una forma útil de trabajar con cuantificadores).

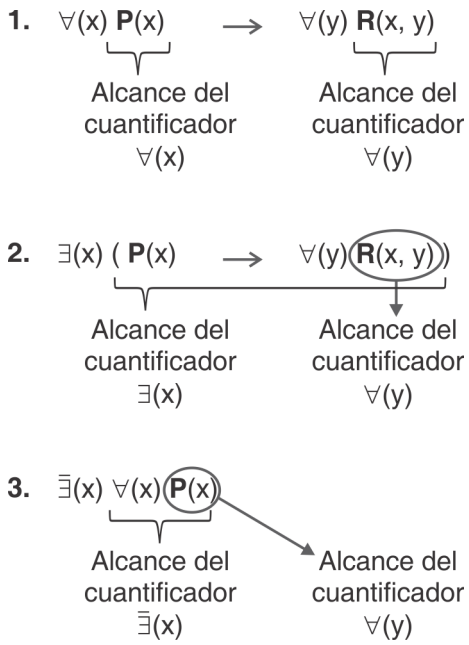
3.3.3. VARIABLES LIBRES Y LIGADAS

En una expresión lógica de la forma $(\forall(x)\ \beta)$ o $(\exists(x)\ \beta)$ se dice que β es el **alcance** o **ámbito** del cuantificador, o sea, es la parte de la expresión a la que se aplica el cuantificador. Para evitar confusiones se debe tener en cuenta los criterios siguientes:

© TODOS LOS DERECHOS RESERVADOS

- Si el cuantificador no va seguido de paréntesis su alcance se restringe a la variable del primer predicado a su derecha.
- Si el cuantificador precede a un paréntesis su alcance corresponde a toda la expresión delimitada por ellos.

A continuación se muestran algunos ejemplos del alcance de un cuantificador:



Se dice que una variable está **ligada** si está en el alcance de algún cuantificador, en caso contrario se dice que está libre. Por ejemplo para $\forall(x)\ P(x, y)$, "x" constituye una variable ligada ya que está bajo la influencia de un cuantificador universal, mientras que la variable "y" es libre pues no se encuentra ligada a ningún cuantificador. Por el contrario en la expresión formalizada $\forall(x)\ \exists(y)\ Q(x, y)$ tanto "x" como "y" representan variables ligadas, "x" por el cuantificador universal, "y" por el cuantificador existencial. En una expresión, una misma variable puede tener ocurrencias libres y ocurrencias ligadas, por ejemplo, en la expresión $\exists(\underline{x})\ P(x) \rightarrow \forall(y)\ R(\underline{x}, y)$, la variable x aparece por primera vez en el alcance del cuantificador $\exists$ (ocurrencia ligada) y luego aparece fuera del alcance de ambos cuantificadores (ocurrencia libre).

En lo referente a las variables ligadas se deben tener en cuenta dos reglas fundamentales:

- Ninguna variable ligada puede estar en el alcance de más de un cuantificador.
- Una variable ligada no puede ser sustituida por constantes.

A modo de conclusión parcial se puede decir que un predicado es un enunciado que al cuantificarse todas sus variables o sustituirse por constantes (elementos bien determinados del dominio) constituye una proposición. Entonces una expresión *A* se denomina **proposición** si y sólo si no contiene variables libres y se denomina **predicado** si y sólo si *A* contiene al menos una variable libre.

© Todos los derechos reservados

3.3.4. NEGACIONES

A menudo hace falta considerar la negación de una expresión cuantificada. Por ejemplo, consideremos la negación del enunciado:

Todos los estudiantes de la clase han cursado una asignatura de Matemática I.

Esta sentencia es una cuantificación universal de la forma $\forall(x)\ P(x)$, donde $P(x)$ es la sentencia "x ha cursado una asignatura de cálculo". La negación de esta sentencia es "No se cumple que todos los estudiantes de la clase hayan cursado una asignatura de Matemática I". Esto es equivalente a "Hay al menos un estudiante en la clase que no ha cursado una asignatura de". Y esto es simplemente la cuantificación existencial de la negación de la función proposicional original, es decir, $\exists(x)\neg \boldsymbol{P}(x)$.

Supongamos que queremos negar una cuantificación existencial. Por ejemplo, considera la proposición "Hay un estudiante en la clase que ha cursado una asignatura de Matemática I".

Ésta es una cuantificación existencial:

$$\exists(x)Q(x),$$

donde *Q(x)* es el predicado "*x* ha cursado una asignatura de Matemática I". La negación de esta sentencia es la proposición "No se cumple que haya un estudiante en la clase que haya cursado una asignatura de Matemática I". Esto es equivalente a "Ninguno de los estudiantes de la clase ha cursado una asignatura de Matemática I", que es justamente la cuantificación universal de la negación de la función proposicional original. Sería equivalente, en lenguaje poco común, a «Para todo estudiante se cumple que no ha cursado un curso de Matemática I, o expresado con cuantificadores,

$$\forall(x)\neg \mathbf{Q}(x)$$

Los cuantificadores combinados con el operador de negación permite representar expresiones como "ningún...", "nadie..." u otra expresión equivalente. Por ejemplo la expresión "Nadie aprobó el examen de suficiencia" se puede formalizar como $\forall(x)\neg P(x)$, donde $\boldsymbol{P}(x)$ representa el predicado "aprobar el examen de suficiencia". La tabla 3.2 muestra algunas expresiones negativas y su formalización.

Tabla 3.2: Combinaciones de los cuantificadores con el operador de negación.

Expresión	Formalización
Ningún x cumple las propiedad P	$\forall(x)\neg \boldsymbol{P}(x)$
Algún x no cumple la propiedad P	$\exists(x)\neg \boldsymbol{P}(x)$
No existe x que no cumplan la propiedad P	$\neg\exists(x)\neg \boldsymbol{P}(x)$
No todos los x cumplen la propiedad P	$\neg\forall(x)\boldsymbol{P}(x)$

© TODOS LOS DERECHOS RESERVADOS

Note que $\neg\forall(x)\boldsymbol{P}(x)$ no significa "nadie" o "ninguna", sino que es falso que todos cumplen con $P(x)$, o sea que existe al menos una x que sí cumple. Sin embargo $\neg\exists(x)\boldsymbol{P}(x)$ sí significa "nadie" ya que expresa que no existe x que cumpla P(x). Por ejemplo decir "Todos en la fiesta se emborracharon" es lo mismo que decir "Es falso que alguien en la fiesta no se emborrachó". Como se puede apreciar existen equivalencias entre el cuantificador existencial y el universal, estas son:

Tabla 3.3: Negación de cuantificadores.

Negación	Fórmula equivalente	¿Cuándo es verdadera la negación?	¿Cuándo es falsa?
$\neg\exists(x)\boldsymbol{P}(x)$	$\forall(x)\neg\boldsymbol{P}(x)$	Para cada *x*. $P(x)$ es falsa.	Hay un *x* para el que $P(x)$ es verdadera.
$\neg\forall(x)\boldsymbol{P}(x)$	$\exists(x)\neg\boldsymbol{P}(x)$	Hay un *x* para el que $P(x)$ es falsa.	$P(x)$ es verdadera para cada *x*.

EJEMPLO 3.3.8.

¿Cuáles son las negaciones de las sentencia $\forall(x)(x^2 > x)$ y $\exists(x)(x^2 = 2)$?

Solución: la negación de $\forall(x)(x^2 > x)$ es la sentencia $\neg\forall(x)(x^2 > x)$, que es equivalente a $\exists(x)\neg(x^2 > x)$. Esto se puede reescribir de la forma $\exists(x)(x^2 \leq x)$. La negación de $\exists(x)(x^2 = 2)$ es $\neg\exists(x)(x^2 = 2)$. La expresión anterior se puede reescribir como $\forall(x)(x^2 \neq 2)$. Los valores de verdad de estas sentencias dependen del dominio.

© TODOS LOS DERECHOS RESERVADOS

3.4. SINTAXIS DE LA LÓGICA DE PREDICADO

Establece los elementos que componen el lenguaje (alfabeto o vocabulario), así como las normas de escritura que permiten relacionarlos correctamente entre sí en correspondencia con el lenguaje de la lógica (gramática).

3.4.1. ALFABETO O VOCABULARIO DE LA LÓGICA DE PREDICADOS

Como se verá a continuación el alfabeto de la Lógica de predicados constituye una extensión del alfabeto proposicional. Los símbolos que conforman el lenguaje de la Lógica de predicado son:

- Constantes proposicionales:

  Son las constantes **0** (falso) y **1** (verdadero) de la Lógica proposicional, utilizadas para denotar el valor de verdad de una proposición.

- Operadores o conectores proposicionales:

  Son los operadores de la Lógica proposicional: *negación* ($\neg$), *disyunción* ($\vee$), *conjunción* ($\wedge$), *implicación* ($\rightarrow$), *bicondicional* ($\leftrightarrow$), utilizados para enlazar predicados.

- Operadores de cuantificación:

  Son los cuantificadores $\forall$, $\exists$ cuyo funcionamiento se explicó en el epígrafe 3.3. Para evitar el uso excesivo de paréntesis se debe tener en cuenta que los cuantificadores y el operador de negación tienen mayor prioridad que el resto de los operadores proposicionales, estos últimos siguen el mismo orden de precedencia que en la Lógica proposicional.

- Constantes individuales:

  Sirven para denotar un elemento único (individuo) dentro del dominio, de ahí el calificativo de individuales. Para su representación se usan generalmente las primeras letras del alfabeto latino en minúsculas; *a, b, c, d* o con subíndices cuando sea necesario a_1, a_2,...También se utilizan símbolos usuales como números: 5, 8; y nombres, *d*: Darién, *f*: Félix.

- Variables individuales:

  Son un conjunto de variables que se utilizan para hacer referencia a un elemento cualquiera del dominio, ejemplo: carro, persona, libro. Comúnmente se utilizan letras minúsculas del final del alfabeto latino, con subíndices de ser necesario, ejemplo: *x, y, z,* x_1, x_2, z_5.

- Funciones n-arias:

  Se representan con letras minúsculas del alfabeto latino *f, g, h,* o estas con subíndices. También se utilizan símbolos comunes como +, –, ·, /. Las funciones reciben como entrada o argumentos elementos del dominio y retornan como valor un solo elemento del dominio. Se utilizan para indicar cómo un elemento queda determinado por otros o para definir nuevos elementos.

- Predicados:

  Las relaciones que se cumplen entre elementos en el dominio constituyen predicados. Se denotan por las letras mayúsculas del alfabeto latino, principalmente P, Q, R, S, o estas con subíndices. También se utilizan símbolos usuales de relación como $\leq$ y $\geq$.

- Signos auxiliares de escritura:

  Como signos auxiliares se emplean los paréntesis y corchetes, con el principal propósito de evitar ambigüedades y la coma para separar elementos.

3.4.2. Gramática. Términos y fórmulas

Las reglas para la construcción de fórmulas en el lenguaje de la lógica de predicados son en su esencia muy similares a las reglas de la lógica proposicional, solo que necesitaremos un nuevo escalón como punto de partida, los términos.

Para la construcción de fórmulas en la Lógica de predicado es necesario formalizar la definición de *término*.

© Todos los derechos reservados

Definición 3.4.1: los ***términos*** son aquellas secuencias de símbolos de la Lógica de predicado que cumplen con las reglas siguientes:

1. Todas las constantes y las variables son términos.
2. Sea f una función n-aria y $t_1, t_2,..., t_n$ términos, entonces $f(t_1, t_2,..., t_n)$ es un término.
3. Todo término es resultado de aplicar las reglas 1 y 2 un número finito de veces.

En otras palabras, las funciones y objetos (constantes y variables) se denominan términos, son expresiones que hacen referencia a un objeto en particular o genérico (persona, lugar, conceptos...), es decir, indican de quién se está hablando, mientras que los predicados expresan de qué se está hablando.

EJEMPLO 3.4.1.

En el listado siguiente se muestran algunos ejemplos donde los términos aparecen subrayados:

- Francisco está de viaje.
- 8 es menor que 13.
- La mesa está sucia.
- El río más largo del mundo es el Amazonas.
- La chica del pelo rubio es modelo.
- El mundial de fútbol de este año está calificado como el más reñido de la historia.

Note que un término no solo hace referencia a un nombre o elemento en particular sino que también puede ser una frase que identifique al objeto particular al que se hace referencia, como es el caso de las tres últimas expresiones.

Luego, una fórmula bien formada se obtiene de aplicar las reglas siguientes:

1. Si P es un predicado n-ario y sean $t_1, t_2,..., t_n$ términos, entonces $P(t_1, t_2,..., t_n)$ es una fórmula bien formada.
2. Si A es una fórmula bien formada, entonces $\neg A$ también lo es.
3. Si A y B son fórmulas bien formadas, entonces $(A \wedge B), (A \vee B), (A \rightarrow B)$ y $(A \leftrightarrow B)$ también lo son.
4. Si A es una fórmula bien formada, donde x es una variable libre, entonces $\forall(x)\ A$ y $\exists(x)\ A$ son fórmulas bien formadas.
5. Toda fórmula bien formada es el resultado de aplicar las reglas 1, 2, 3 y 4 un número finito de veces.

© TODOS LOS DERECHOS RESERVADOS

Los símbolos A y B usados para definir las reglas de formación de fórmulas no pertenecen al alfabeto, son símbolos del lenguaje natural utilizados para hacer referencia a cualquier fórmula del lenguaje de la Lógica de predicado.

EJEMPLO 3.4.2.

Determine si las siguientes constituyen fórmulas bien formadas (FBF):

a) $5 \cdot b + 2 \geq a$

b) $\exists(z)\ (x + y + 1 = z)$

c) $\neg\forall\exists(x)P(x)$

d) $P(f(x), y) \rightarrow Q(z, y)$

e) $\forall(x)\ \forall(y)\ (x + y = a)$

f) $\exists(x)\ x\ P$

g) $\exists(x)\ P(x) \vee Q\ f(x)a$

Solución:

a) Según la regla 1 de los términos, *a* y *b* son constantes por lo que constituyen términos, mientras que la regla 2 garantiza que "$5 \cdot b$", en lo adelante t_1, es también un término, donde se utiliza la notación infija para la función "·". Igualmente "t_1 + 2", en lo adelante t_2, es un término con notación infija en la función "+". Luego utilizando la regla 1 de formación de fórmulas "$t_2 \geq a$" es una FBF, que también utiliza notación infija. Como se puede apreciar "$5 \cdot b + 2 \geq a$" es una FBF.

b) La regla 1 de los términos garantiza que *x, y, z* son términos. Luego según la segunda regla "$x + y$", en lo adelante t_1, es también un término y por tanto "$t_1 + 1$". Para mayor comodidad el término "$t_1 + 1$" será denominado t_2. Con la regla 1 de formación de fórmula se puede comprobar que "$t_2 = z$", es una FBF. Como *z* es una variable libre en la fórmula $(x + y + 1 = z)$ entonces se cumple la regla 3 por lo que "$\exists(z)\ (x + y + 1 = z)$" es una FBF.

c) Sea "*P*" un predicado y "*x*" un término, entonces según la regla 1, "$P(x)$" es una FBF. Siguiendo la regla 3, "$\exists\ (x)\ P(x)$" una FBF. Si se sustituye "$\exists\ (x)\ P(x)$" por el símbolo A_0 quedaría como "$\neg\forall A_0$" lo que incumple con la regla 3 por lo que el inciso *c* no constituye una FBF.

d) *x, y, z* constituyen variables por lo que son consideradas términos aplicando la regla 1 para los términos, mientras que la regla 2 asegura que la función "*f(x)*" es también un término. Entonces sean *P* y *Q* predicados binarios, con la regla 1 de formación de fórmulas tanto "$P(f(x), y)$" como "$Q(z, y)$" son FBF. Sustituyendo "$P(f(x), y)$" por *A* y "$Q(z, y)$" por *B*, se puede observar que $A \rightarrow B$ cumple con la regla 3 por lo que "$P(f(x),y) \rightarrow Q(z, y)$" es una FBF.

e) Siguiendo el mismo análisis realizado en el inciso b para $(x + y + 1 = z)$ se puede afirmar que $(x + y = a)$, denominada "*A*" en lo adelante, es una FBF, siendo "*x*", "*y*" variables y "*a*" una constante. Como "*y*" es una variable libre dentro la fórmula *A*

© TODOS LOS DERECHOS RESERVADOS

entonces, aplicando la regla 4, "$\forall(y)\ (x + y = a)$" es una FBF y por tanto, ya que "x" también es una variable libre, "$\forall(x)\ \forall(y)\ (x + y = a)$" es una FBF.

f) "$x\ P$" no es una FBF pues no cumple con ninguna de las reglas establecidas, por tanto la fórmula "$\exists(x)\ x\ P$" no es una FBF.

g) Como se explicó en el inciso *c*, "$P(x)$" es una FBF, por tanto según la regla 4, "$\exists(x)\ P(x)$" también lo es. Sin embargo "$Q\ f(x)a$" incumple la regla 1 de formación de fórmula por lo que "$\exists(x)\ P(x) \vee Q\ f(x)a$" no constituye una FBF

3.5. FORMALIZACIÓN DE EXPRESIONES DEL LENGUAJE NATURAL EN LÓGICA DE PREDICADOS

Traducir frases del lenguaje natural a expresiones lógicas es una tarea crucial en matemáticas, programación lógica, inteligencia artificial, ingeniería del software y muchas otras disciplinas. Comenzamos estudiando esta cuestión en el capítulo 1, donde utilizamos fórmulas para expresar afirmaciones en expresiones lógicas. En ese momento evitamos a propósito afirmaciones cuya traducción requiriese predicados y cuantificadores. Formalizar expresiones del lenguaje natural en expresiones lógicas se hace más complejo cuando, se necesitan cuantificadores. Además, puede haber diferentes formas de traducir una frase particular. (En consecuencia, no hay un «libro de recetas» que se pueda seguir paso a paso). Utilizaremos algunos ejemplos para ilustrar cómo se formalizan afirmaciones del lenguaje natural en lógica de predicados. El objetivo es producir expresiones lógicas simples y útiles.

© Todos los derechos reservados

Ejemplo 3.5.1.

Expresa la frase «*Todo estudiante de esta clase ha estudiado programación*» utilizando predicados y cuantificadores.

Solución: la expresión original se puede reescribir como «*Para todo estudiante de esta clase, ese estudiante ha estudiado programación*».

Luego introducimos la variable x, de tal forma que nuestra sentencia se convierte en «*Para todo estudiante x de esta clase, x ha estudiado programación*».

Continuando, introducimos el predicado $C(x)$, que es el enunciado «*x ha estudiado cálculo*». Por consiguiente, si el dominio de x consiste en los estudiantes de la clase, podemos traducir nuestra frase como $\forall(x)C(x)$.

Podemos estar interesados en un grupo de personas más amplio que el de esa clase en particular. Si cambiamos el dominio, tomando el conjunto de todas las personas, habremos de expresar nuestro enunciado como:

«*Para toda persona x, si la persona x es un estudiante de esta clase, entonces x ha estudiado cálculo*».

Si $S(x)$ representa el predicado de que la persona x está en la clase, nuestra sentencia se puede expresar como $\forall(x)[S(x) \rightarrow C(x)]$.

Es importante puntualizar que nuestra sentencia no puede expresarse como $\forall(x)[S(x) \wedge C(x)]$, puesto que esta sentencia afirmaría que todas las personas son estudiantes de la clase y han estudiado cálculo.

Finalmente, cuando estamos interesados en los estudios de una persona, aparte de la programación, podríamos preferir usar un cuantificador de dos variables $Q(x, y)$ para la frase «el estudiante x ha estudiado la asignatura y». Así, podríamos reemplazar $C(x)$ por $Q(x,$ *cálculo*) en las dos opciones que hemos elegido para obtener $\forall(x)C(x,$ *cálculo*) o $\forall(x)[S(x) \rightarrow C(x,$ *cálculo*$)]$.

Ejemplo 3.5.2.

Formalice las expresiones siguientes:

a) Algunos no tienen amigos.

b) No todas las personas que saben manejar tienen automóvil.

c) Si todos los amigos de Juan saben programar a Juan le gusta la informática.

Solución:

a) Tomando como predicado $P(x, y)$: "x es amigo de y" quedaría la fórmula $\exists(x)\neg\exists(y)P(x, y))$.

b) Sea $Q(x)$: "x sabe manejar" y $R(x)$: "x tiene automóvil", aplicando lo estudiado en el capítulo de Lógica de predicado, la fórmula $Q(x) \wedge R(x)$ permite expresar que "las personas que saben manejar tienen automóvil" ya que la conjunción obliga a que $Q(x)$ y $R(x)$ sean verdaderas para cumplirse. Luego al introducir un cuantificador universal negativo se indica que no todas las "x" cumplen lo expresado, con lo que se obtiene finalmente la fórmula $\neg\forall(x)(Q(x)\wedge R(x))$. También se puede expresar como $\exists(x)\neg[Q(x) \wedge R(x)]$ según las equivalencias entre cuantificadores vistas anteriormente.

c) Se tomarán como predicados $S(x)$: "a x le gusta la informática", $T(x)$: "x es amigo de y", $U(x)$: "x sabe programar" y a "j" como la constante que identifica el elemento Juan. A través de la fórmula $T(x, j)$ se hace alusión a "los amigos de Juan", mientras que $(T(x, j) \wedge U(x))$ se refiere a "los que son amigos de Juan y saben programar". Cuantificando la variable x en esta última expresión se obtiene que "todos los amigos de Juan saben programar". Luego la expresión final sería $\forall(x)\ [T(x, j) \wedge U(x)] \rightarrow S(j)$, donde $S(j)$ expresa que "a Juan le gusta la informática".

© Todos los derechos reservados

EJEMPLO 3.5.3.

Exprese en el lenguaje de la lógica los siguientes enunciados:

a) Para toda x y para toda y existe una z, todos reales, tal que $x + y = z$ o $y = z + 1$.

b) Existe algún médico amigo de Teresa.

Solución:

a) Dominio= $\{x \mid x \in R\}$

$I(x, y)$: $x = y$

$\forall(x)\forall(y)\exists(z)[I(x + y, z) \vee I(y, z + 1)]$

b) Dominio= {personas}

M(x): x es médico

$G(x, y)$: x es amigo de y

t: Teresa

$\exists(x)[M(x) \rightarrow G(x, t)]$

3.5.1. CUANTIFICADORES ANIDADOS

Ya definimos los cuantificadores universal y existencial y mostramos cómo se pueden usar en la construcción de sentencias matemáticas. También explicamos cómo se pueden utilizar para formalizar frases en lenguaje natural, convirtiéndolas en expresiones lógicas. En esta sección estudiaremos cuantificadores anidados, que son cuantificadores que se localizan dentro del rango de aplicación de otros cuantificadores, como en la sentencia $\forall(x)\exists(y)[P(x, y) \rightarrow Q(y, x)]$. Los cuantificadores anidados se usan tanto en matemáticas como en ciencias de la computación. Aunque a veces pueden ser difíciles de entender, las reglas que ya hemos estudiado nos ayudarán a trabajar con ellos.

En muchos contextos aparecen complicadas expresiones que hacen uso de cuantificadores. Para entender estas sentencias con muchos cuantificadores, debemos desenmarañar el significado de cada cuantificador y predicado que aparecen.

EJEMPLO 3.5.4.

Traduce a lenguaje natural la sentencia:

$$\forall(x)\forall(y)[(x > 0) \wedge (y < 0) \rightarrow (xy < 0)]$$

© TODOS LOS DERECHOS RESERVADOS

donde el dominio para ambas variables consiste en los números reales.

Solución: esta sentencia afirma que para todo par de números reales x e y, si $x > 0$ e $y < 0$, entonces $xy < 0$. Esto es, que para los pares de números reales x e y, si x es positivo e y es negativo, entonces xy es negativo. Esto se puede afirmar más sucintamente como "*El producto de un número real positivo y un número real negativo es un número real negativo*".

Las expresiones con cuantificadores anidados que formulan sentencias en lenguaje natural pueden ser muy complicadas. El primer paso para traducir esas expresiones es escribir qué expresan los cuantificadores y predicados de la expresión. El siguiente paso es expresar el significado en una frase sencilla.

Ejemplo 3.5.5.

Traduce la sentencia:

$$\forall(x)[C(x) \vee \exists(y)[C(y) \wedge F(x, y)]]$$

a lenguaje natural, donde $C(x)$ "es x tiene un ordenador", $F(x, y)$ es "x e y son amigos" y el dominio tanto para x como y consiste en todos los estudiantes de tu facultad.

Solución: la sentencia afirma que para cada estudiante x de tu facultad, x tiene un ordenador o hay un estudiante y tal que y tiene un ordenador y x e y son amigos. En otras palabras, todo estudiante de tu facultad tiene un ordenador o un amigo que tiene uno.

Ejemplo 3.5.6.

Traduce la sentencia:

$$\exists(x)\forall(y)\forall(z)[[F(x, y) \wedge F(x, z) \wedge (y \neq z)] \rightarrow \neg F(y, z)]$$

a lenguaje natural, donde $F(a, b)$ significa que a y b son amigos. El dominio para x, y y z consiste en todos los estudiantes de tu facultad.

Solución: primero examinamos la expresión $[F(x, y) \wedge F(x, z) \wedge (y \neq z)] \rightarrow \neg F(y, z)$. Esta expresión dice que si los estudiantes x e y son amigos, los estudiantes x y z son amigos y, además, y y z no son la misma persona, entonces y y z no son amigos. Se sigue que la sentencia original, triplemente cuantificada, dice que hay un estudiante x tal que para todos los estudiantes y y todos los estudiantes z que son diferentes de y, si x e y son amigos y x y z también son amigos, entonces y y z no son amigos. En otras palabras, hay un estudiante para el cual se cumple que sus amigos no son amigos entre sí.

© Todos los derechos reservados

EJEMPLO 3.5.7.

Expresa la sentencia "*Si una persona es del sexo femenino y tiene un hijo, esta persona es la madre de alguien*" como una expresión lógica que involucre predicados, cuantificadores cuyo dominio es el conjunto de todas las personas y conectivos lógicos.

Solución: la frase anterior se puede expresar como < Para toda persona x, si la persona x es del sexo femenino y la persona x tiene un hijo, entonces existe una persona y tal que la persona x es madre de la persona y. Introducimos los predicados $F(x)$ para representar < a es del sexo femenino*, $P(x)$ para representar x tiene un hijo y $M(x, y)$ para representar "x es madre de y". La frase original se puede expresar como $\forall(x)[[F(x) \wedge P(x)] \rightarrow \exists y M(x, y)]$ obteniendo una expresión equivalente de la siguiente forma:

$$\forall(x)\exists y[[F(x) \wedge P(x)] \rightarrow M(x, y)]$$

EJEMPLO 3.5.8.

Expresa la sentencia "*Cada persona tiene exactamente un amigo preferido*" como una expresión lógica que involucre predicados, cuantificadores –cuyo dominio es el conjunto de todas las personas– y conectivos lógicos.

Solución: la frase anterior se puede expresar como "Para cada persona x, la persona x tiene exactamente un amigo preferido". Introduciendo el cuantificador universal, se ve que la sentencia es la misma que "$\forall(x)$ (la persona x tiene exactamente un amigo preferido)", donde el dominio corisiste en toda la gente.

Por otra parte expresar que x tiene exactamente un amigo preferido significa que hay una persona y que es el mejor amigo de x, y además, que para toda persona z, si z no es la persona y, entonces z no es el mejor amigo de x. Cuando introducimos el predicado $A(x, y)$ como "y es el mejor amigo de x", la sentencia que afirma que x tiene exactamente un amigo preferido se puede representar como:

$$\exists(y)\ [A(x, y) \wedge \forall(z)[(z \neq y) \rightarrow \neg A(x, z)]].$$

EJEMPLO 3.5.9.

Formalice la siguiente expresión: "*Hay una mujer que ha viajado en un vuelo en cada una de las líneas aéreas del mundo*".

Solución:

$P(x, v)$: x ha viajado en y.

© TODOS LOS DERECHOS RESERVADOS

$Q(v, z)$: v es un vuelo de la línea aérea z.

Donde los dominios para x, v y z consisten en todas las mujeres, todos los vuelos y todas las líneas aéreas, respectivamente.

Por lo que podemos expresar la sentencia dada como:

$$\exists(x)\forall(z)\exists(v)[P(x, v) \wedge Q(v, z)],$$

manteniendo el mismo dominio de discurso podríamos a ver expresado la sentencia de la siguiente manera:

$$\exists(x)\forall(z)\exists(v)R(x, v, z),$$

donde $R(x, v, z)$: x ha viajado en el vuelo v de la línea aérea z.

Ejemplo 3.5.10.

Formaliza la afirmación "*Todo número real, excepto el cero, tiene un inverso para el producto*".

Solución: primero reescribimos la frase como «Para todo número real x, excepto el cero, x tiene un inverso para el producto». Esta sentencia se puede reescribir de nuevo como «Para todo número real x, si $x \neq 0$, entonces existe un número real y tal que $xy = 1$. Esto se puede expresar como:

$$\forall(x)[(x \neq 0) \rightarrow \exists y(xy = 1)].$$

Ejemplo 3.5.11.

Usa cuantificadores para expresar la sentencia "*No existe ninguna mujer que haya viajado en un vuelo de cada una de las líneas aéreas del mundo*".

Solución: la afirmación anterior es la negación de la sentencia «Hay una mujer que ha viajado en un vuelo de cada línea aérea del mundo» del Ejemplo 3.5.9, donde se formalizaba como $\exists(x)\forall(z)\exists(v)[P(x, v) \wedge Q(v, z)]$. Por lo que nuestra sentencia se puede expresar como $\neg\exists(x)\forall(z)\exists(v)[P(x, v) \wedge Q(v, z)]$. Aplicando sucesivamente las reglas de la negación de sentencias cuantificadas para desplazar la negación dentro de cada cuantificador y aplicando las leyes de De Morgan en el último paso, encontramos que nuestra sentencia es equivalente a cada una de las de la siguiente secuencia:

© Todos los derechos reservados

$$\forall(x)\neg\forall(z)\exists(v)[P(x, v) \wedge Q(v, z)] \equiv \forall(x)\exists(z)\neg\exists(v)[P(x, v) \wedge Q(v, z)]$$

$$\equiv \forall(x)\exists(z)\forall(v)\neg[P(x, v) \wedge Q(v, z)]$$

$$\equiv \forall(x)\exists(z)\forall(v)[\neg P(x, v) \vee \neg Q(v, z)].$$

Pensando en los cuantificadores como bucles:

Al trabajar con cuantificadores de más de una variable es útil a veces pensar en términos de bucles anidados. (Por supuesto, si hay un número infinito de elementos en el dominio de alguna variable, no podemos cerrar un bucle para todos los valores. A pesar de ello, esta forma de pensar sigue siendo útil). LA ideas que a continuación se exponen nos permitirán ver la relación estrecha entre los bucles estudiados en programación y los cuantificadores propiamente dichos. Por ejemplo, para ver si $\forall(x)\forall(y)$ P(x, y) es verdadera, recorremos en un bucle todas las variables x, y para cada x recorremos en un segundo bucle todos valores de y. Si encontramos que $P(x, y)$ es verdadera para todos los valores de x e y, hemos determinado que $\forall(x)\forall(y)$ $P(x, y)$ es verdadera. Si, por el contrario, encontramos algún valor de x para el cual hay un valor de y tal que $P(x, y)$ resulta ser falsa, hemos demostrado que $\forall(x)\forall(y)$ $P(x, y)$ es falsa.

De forma similar, para determinar si $\forall(x)\exists(y)P(x, y)$ es verdadera, recomemos en un bucle todos los valores de x. Para cada x, recorremos en un bucle los valores de y hasta que encontramos un y para el cual $P(x, y)$ es verdadera. Si para todos los x encontramos tal valor de y, entonces $\forall(x)\exists(y)P(x, y)$ es verdadera; si para algún x no encontramos un valor de y con esa propiedad, entonces $\forall(x)\exists(y)P(x, y)$ es falsa.

Para ver si $\exists(x)\forall(y)P(x, y)$ es verdadera, recorremos los valores de x en un bucle hasta que encontramos un x para el cual $P(x, y)$ es siempre verdadera cuando recorremos en un bucle todos los valores de y. Una vez encontrado tal valor de x, sabemos que $\exists(x)\forall(y)P(x, y)$ es verdadera. Si no encontramos nunca un x como ése, entonces sabremos que $\exists(x)\forall(y)P(x, y)$ es falsa.

Finalmente, para saber si $\exists(x)\forall(y)P(x, y)$ es verdadera, recorremos en un bucle los valores de x, y para cada valor de x recorremos los valores de y hasta que encontremos un x para el cual haya un y que verifique que $P(x, y)$ sea verdadera.

Ejemplo 3.5.12.

Sea $Q(x, y, z)$ la sentencia $x + y = z$. ¿Cuáles son los valores de verdad de las sentencias $\forall(x)\forall(y)\exists(z)Q(x, y, z)$ y $\exists(z)\forall(x)\forall(y)Q(x, y, z)$?

Solución: supongamos que asignamos valores a x e y. Entonces, existe un número real z tal que $x+y=z$. Por consiguiente, la cuantificación $\forall(x)\forall(y)\exists(z)Q(x, y, z)$ que es la sentencia:

"Para todos los números reales x e y hay un número real z tal que x + y = z"

es verdadera. El orden de la cuantificación aquí importa, ya que:

© Todos los derechos reservados

$\exists(z)\forall(x)\forall(y)Q(x, y, z)$ es la sentencia:

"Hay un número real z tal que para todos los números des x e y se cumple que x + y =z"

la cual es falsa, ya que ningún valor de z satisface la ecuación $x + y = z$ para todos los valores de x e y.

En la tabla 3.4 aparece un resumen con los significados de las diferentes cuantificaciones posibles con dos variables.

Tabla 3.4: Resumen cuantificadores de dos variables.

Sentencia	¿Cuándo es verdadera?	¿Cuándo es falsa?
$\forall(x)\forall(y)P(x, y)$ $\forall(y)\forall(x)P(x, y)$	$P(x, y)$ es verdadera para todo par x, y.	Hay un par x, y para el cual $P(x, y)$ es falsa.
$\forall(x)\exists(y)P(x, y)$	Para todo x hay un y para el cual $P(x, y)$ es verdadera.	Hay un x tal que $P(x, y)$ es falsa para todo y.
$\exists(x)\forall(y)P(x, y)$	Hay un x para el cual $P(x, y)$ es verdadera para todo y.	Para todo x hay un y para el cual $P(x, y)$ es falsa.
$\exists(x)\exists(y)P(x, y)$ $\exists(y)\exists(x)P(x, y)$	Hay un par x, y para el cual es $P(x, y)$ verdadera.	$P(x, y)$ es falsa para todo par x, y.

© Todos los derechos reservados

3.6. SEMÁNTICA DE LA LÓGICA DE PREDICADO

La semántica se encarga de estudiar la relación de los signos con su significado, es decir, atribuye significados a las distintas fórmulas del lenguaje, utilizando para esto las tablas de verdad.

3.6.1. Interpretación de fórmulas

Ya sabemos cómo formular un enunciado del lenguaje natural en el lenguaje de la lógica de predicados, veamos entonces cómo determinar su valor de verdad según determinadas reglas de evaluación para cada elemento involucrado en la fórmula.

En la lógica proposicional vimos que una interpretación consiste en una asignación de valores tomados de un conjunto de valores atribuibles a las variables proposicionales que ocurren en la fórmula en cuestión.

En las fórmulas de la lógica de predicados ocurren variables que no necesariamente toman valores en un conjunto de valores atribuibles que determinan valores de verdad, sino que están definidas en un cierto dominio, que puede ser finito o infinito.

Es indispensable para poder evaluar una interpretación de una fórmula que contiene cuantificadores, tener bien determinado el dominio de discurso de las variables cuantificadas.

De modo que en dependencia del dominio que se escoja, una fórmula podría ser verdadera o no.

Como en la Lógica proposicional, es necesario saber si una fórmula es verdadera o falsa, es decir, realizar una interpretación de la fórmula. Para definir una interpretación de una fórmula de la Lógica de predicado se debe construir una estructura adecuada como mecanismo que permita asignar un significado a cada símbolo de la fórmula.

Una estructura adecuada para la interpretación de una fórmula consiste en:

- Definir un conjunto no vacío U, utilizado como dominio de discurso.
- Definir para cada variables libres x el dominio de discurso U.
- Asignar a cada constante de la fórmula un elemento del dominio U.
- Evaluar cada función n-aria f en el dominio U^n, f denota una función con dominio en U y valores en U.
- Evaluar cada una de las proposiciones que se obtienen en el dominio U a partir de un predicado P.

El conjunto de significados y las reglas de evaluación de los operadores proposicionales son los mismos utilizados en la Lógica proposicional.

Para asignar un valor de verdad a expresiones con cuantificadores se sigue el procedimiento explicado en el epígrafe 3.3 que se resume como sigue:

Si A es de la forma $\forall(x)\ P(x)$ y $V_I(A)$ representa su valor de verdad bajo la interpretación I, entonces:

$$V_I(A) = \begin{cases} \mathbf{1} \text{ si } P(u) = \mathbf{1} \text{ para todo } u \in U \\ \\ \mathbf{0} \text{ en el caso contrario} \end{cases}$$

Si A es de la forma $\exists(x)\ P(x)$ y $V_I(A)$ representa su valor de verdad bajo la interpretación I, entonces:

$$V_I(A) = \begin{cases} \mathbf{1} \text{ si } P(u) = 1 \text{ para algún } u \in U \\ \\ \mathbf{0} \text{ en caso contrario} \end{cases}$$

Nota: En ambos casos $P(u)$ se obtiene de sustituir en la fórmula $P(x)$ todas las ocurrencias de x por u, donde u representa cada elemento del dominio.

© Todos los derechos reservados

EJEMPLO 3.6.1.

Interpretación de fórmulas.

a) Dada la fórmula $\forall(x)\ \exists(y)\ (P(x, y) \vee Q(f(x)))$ determine su valor de verdad bajo la interpretación siguiente:

- Dominio $U = \{2, 3, 4, 5\}$
- Función $f(x) = (2 \cdot x) - 1$
- Predicados $P(x, y)$: "$x \leq 2 \cdot y$"
  $Q(x)$: "es número primo"

b) Sea $P(x, y)$ el predicado "x es más viejo que y". El dominio de discurso consta de tres vecinos: Lucía de 28 años, Juan de 47 años, Isabel de 43 años y Pedro de 30 años. Formalice la expresión "*Isabel no es la más vieja*" y determine su valor veritativo.

Solución:

a) Según los cuantificadores usados se debe verificar que para cada "x" del dominio U existe una "y" para la cual "$P(x, y) \vee Q(f(x))$" es verdadera.

Se comienza con $x = 2$:

Para y = 2 sería:

$$\underbrace{\underbrace{P(2, 2)}_{\text{"}2 \leq 4\text{"}}}_{1} \ \mathbf{v}\ \underbrace{\underbrace{Q(\underbrace{f(2)}_{\text{"}3\text{"}})}_{\text{"3 es número primo"}}}_{1}$$

$$1 \ \mathbf{v}\ 1$$

$$1$$

Para y = 3

$$\underbrace{\underbrace{P(2, 3)}_{\text{"}2 \leq 6\text{"}}}_{1} \ \mathbf{v}\ \underbrace{\underbrace{Q(\underbrace{f(2)}_{\text{"}3\text{"}})}_{\text{"3 es número primo"}}}_{1}$$

$$1 \ \mathbf{v}\ 1$$

$$1$$

Para y = 4

$$\underbrace{\underbrace{P(2, 4)}_{\text{"}2 \leq 8\text{"}}}_{1} \ \mathbf{v}\ \underbrace{\underbrace{Q(\underbrace{f(2)}_{\text{"}3\text{"}})}_{\text{"3 es número primo"}}}_{1}$$

$$1 \ \mathbf{v}\ 1$$

$$1$$

Para y = 5

$$\underbrace{\underbrace{P(2, 5)}_{\text{"}2 \leq 10\text{"}}}_{1} \ \mathbf{v}\ \underbrace{\underbrace{Q(\underbrace{f(2)}_{\text{"}3\text{"}})}_{\text{"3 es número primo"}}}_{1}$$

$$1 \ \mathbf{v}\ 1$$

$$1$$

© TODOS LOS DERECHOS RESERVADOS

Para $x = 2$ se cumple la fórmula, por lo que se debe pasar al siguiente elemento del dominio $x = 3$.

Para y = 2

$$\underbrace{\underbrace{P(3, 2)}_{\text{“}3 \leq 4\text{”}}}_{1} \;\mathbf{v}\; \underbrace{\underbrace{Q(\underbrace{f(3)}_{\text{“}5\text{”}})}_{\text{“5 es número primo”}}}_{1}$$

$$1$$

Si “x” es menor e igual que “$2 \cdot y$” para $y = 2$, entonces también lo es para las $y > 2$, por lo que $P(3, y)$ es verdadera para toda “y” en el dominio U, es decir, $P(3, 3)$, $P(3, 4)$ y $P(3, 5)$ son verdaderas. Luego como $Q(f(3))$ es verdadera, entonces:

- $P(3, 3) \vee Q(f(3))$ es verdadera.
- $P(3, 4) \vee Q(f(3))$ es verdadera.
- $P(3, 5) \vee Q(f(3))$ es verdadera.

Para $x = 3$ la fórmula es verdadera, por lo que se debe pasar al siguiente elemento del dominio $x = 4$.

Para y = 2

$$\underbrace{\underbrace{P(4, 2)}_{\text{“}4 \leq 4\text{”}}}_{1} \;\mathbf{v}\; \underbrace{\underbrace{Q(\underbrace{f(4)}_{\text{“}7\text{”}})}_{\text{“7 es número primo”}}}_{1}$$

$$1$$

Ya que Si “x” es menor e igual que “$2 \cdot y$” para $y = 2$, entonces también lo es para las $y > 2$, entonces:

- $P(4, 3) \vee Q(f(4))$ es verdadera.
- $P(4, 4) \vee Q(f(4))$ es verdadera.
- $P(4, 5) \vee Q(f(4))$ es verdadera.

Para $x = 4$ la fórmula es verdadera, por lo que se debe pasar al siguiente elemento del dominio $x = 5$.

© Todos los derechos reservados

Para y = 2

$$\underbrace{\underbrace{P(5,2)}_{\text{“}5 \leq 4\text{”}}}_{0} \;\vee\; \underbrace{\underbrace{Q(\underbrace{f(5)}_{\text{“}9\text{”}})}_{\text{“9 es número primo”}}}_{0}$$

$$\underbrace{0 \;\vee\; 0}_{0}$$

Al encontrar un valor de “*y*” en el dominio de discurso para el cual la fórmula “$P(x, y) \vee Q(f(x))$” es falsa, se detiene el proceso, concluyendo que la fórmula “$\forall(x)\ \exists(y)\ P(x, y) \vee Q(f(x))$” es falsa bajo esta interpretación.

b) La expresión “*Isabel no es la más vieja*” se puede reformular como “Existen vecinos más viejos que Isabel” que en el lenguaje de la Lógica de predicado sería $\exists(x)\ P(x, i)$, donde “*i*” es la constante que identifica el elemento “Isabel” dentro del dominio de vecinos. Para afirmar que $\exists(x)\ P(x, i)$ es verdadera basta con encontrar en el dominio un vecino “*x*” con más edad que Isabel, en caso contrario es falsa. En el proceso de verificación se utilizarán las constantes *l*: “Lucía”, *j*: “Juan” y *p*: “Pedro”

- Para *x* = “Lucía”, $P(l, i)$ es falsa por lo que se pasa al otro elemento del dominio.
- Para x = “Juan”, $P(j, i)$ es verdadera. Como se encontró una x que hace verdadera la fórmula se detiene la búsqueda y se concluye que $\exists(x)\ P(x, i)$ es verdadera para la interpretación dada.

En la Lógica proposicional se manejó el concepto de satisfacibilidad, extrapolándolo a la Lógica de predicados, una fórmula *A* es **satisfacible** si existe alguna interpretación para la cual *A* tenga valor veritativo “1”.

3.7. LEYES DE LA LÓGICA PREDICADOS

Con el lenguaje de la Lógica de Predicados las leyes de la lógica se enriquecen con nuevas equivalencias producto de la inclusión de los cuantificadores. En el ámbito de la Lógica de predicado, se dice que dos fórmulas *A* y *B* son equivalentes si sus valores veritativos son iguales en cualquier interpretación. Algunas de estas equivalencias se listan a continuación:

Leyes de Morgan para cuantificadores negativos:

1. $\neg\forall(x)\ \boldsymbol{P}(x) \equiv \exists(x)\ \neg\boldsymbol{P}(x)$
2. $\neg\exists(x)\ \boldsymbol{P}(x) \equiv \forall(x)\ \neg\boldsymbol{P}(x)$

© Todos los derechos reservados

Cambio de variables:

3. $\forall(x)\ \boldsymbol{P}(x) \equiv \forall(y)\ \boldsymbol{P}(y)$

Conmutativa:

4. $\forall(x)\ \forall(y)\ \boldsymbol{P}(x, y) \equiv \forall(y)\ \forall(x)\ \boldsymbol{P}(x, y)$
5. $\exists(x)\ \exists(y)\ \boldsymbol{P}(x, y) \equiv \exists(y)\ \exists(x)\ \boldsymbol{P}(x, y)$
6. $\exists(y)\forall(x)\ \boldsymbol{P}(x, y) \rightarrow \forall(y)\ \exists(x)\ \boldsymbol{P}(x, y)$ * La implicación se cumple solo en el sentido indicado.

Disyunción:

7. $\exists(x)\ (\boldsymbol{A}(x) \vee \boldsymbol{B}(x)) \equiv \exists(x)\boldsymbol{A}(x) \vee \exists(x)\boldsymbol{B}(x)$
8. $\forall(x)\ \boldsymbol{A}(x) \vee \forall(x)\ \boldsymbol{B}(x) \rightarrow \forall(x)\ (\boldsymbol{A}(x) \vee \boldsymbol{B}(x))$ * La implicación se cumple solo en el sentido indicado.

Conjunción:

9. $\forall(x)\ (\boldsymbol{A}(x) \wedge \boldsymbol{B}(x)) \equiv \forall(x)\boldsymbol{A}(x) \wedge \forall(x)\boldsymbol{B}(x)$
10. $\exists(x)\ (A(x) \wedge \boldsymbol{B}(x)) \rightarrow \exists(x)\boldsymbol{A}(x) \wedge \exists(x)\boldsymbol{B}(x)$ * La implicación se cumple solo en el sentido indicado.

© Todos los derechos reservados

Resumen

© Todos los derechos reservados

© Todos los derechos reservados

Estructuras deductivas 4

Objetivos

- Analizar el concepto y empleo de las estructuras deductivas.
- Caracterizar las estructuras deductivas.
- Inferir una conclusión a partir de un conjunto de premisas, utilizando las leyes y reglas de inferencias.
- Caracterizar las técnicas de demostración: directa, por contradicción y por inducción matemática.
- Aplicar las reglas de inferencia y la inducción matemática en la realización de demostraciones.

© Todos los derechos reservados

4.1. INTRODUCCIÓN

Todo proceso de razonamiento, principalmente los del tipo deductivo, pueden ser formalizados a través de estructuras deductivas. Como se verá en este capítulo las estructuras deductivas son la base fundamental de las demostraciones en el campo de las matemáticas. En este mismo contexto se encuentran las inferencias estadísticas que permiten la inferencia de datos cuantitativos y el cálculo de la probabilidad de una conclusión y de alternativas. Las estructuras deductivas tienen gran aplicación en el área de la inteligencia artificial para la construcción de sistemas de expertos, donde se parte de una base de conocimiento (premisas) que puede ser extendida automáticamente con el uso de reglas de inferencia, permitiendo así el aprendizaje autónomo del sistema, emulando el pensamiento humano. También se utilizan en ciencias de la computación para verificar el correcto funcionamiento de los programas; en las ciencias físicas y naturales, son útiles al realizar conclusiones de experimentos; y en la vida cotidiana se emplean casi intuitivamente en disímiles situaciones.

En este capítulo se realiza un análisis de las estructuras deductivas, su formulación y la determinación de la validez semántica de los razonamientos deductivos representados mediante estas, así como un conjunto de leyes y reglas de inferencia que permiten demostrar estructuras más complejas. En adición a lo antes planteado se presentan diferentes métodos de demostración matemática que en su esencia implican el uso de las reglas de inferencia antes mencionadas.

4.2. ESTRUCTURAS DEDUCTIVAS

Los seres humanos tienen la capacidad de razonar, proceso mediante el cual se enlazan ideas de las que surgen otras. Todo razonamiento se basa en un conjunto de **premisas** que representan el conocimiento adquirido que al procesarlas permiten arribar a una **conclusión**, un nuevo conocimiento. Existen varias formas de razonamiento: deductivo, inductivo y el abductivo.

- Deductivo: es aquel razonamiento en el cual se parte de lo universal hacia lo particular, es decir, a partir de las causas se infieren los efectos. Ejemplo:

  Premisas:

  p: Los animales que vuelan tienen alas.

  q: Las palomas vuelan.

  Conclusión:

  r: Las palomas tienen alas.

- Inductivo: a diferencia del deductivo se parte de premisas de datos particulares para obtener conclusiones de carácter general. Ejemplo:

  Premisas:

  q: Las palomas vuelan.

  r: Las palomas tienen alas.

  Conclusión:

  p: Los animales que vuelan tienen alas.

  La generalización no garantiza que la conclusión sea correcta, en el ejemplo, se asume que todos los animales que vuelan tienen alas a partir del conocimiento de que las palomas vuelan y tienen alas. Sin embargo, teóricamente se podría pensar en la posibilidad de que existiera un animal que volara sin necesidad de tener alas, pues solo se tiene la certeza de que esta conclusión se cumple para las palomas. Los mecanismos para certificar que una generalización es correcta se encuentran en el campo de la inducción matemática.

- Abductivo: en el caso de la abducción se trata de explicar la conclusión considerando las premisas como hipótesis explicativas. Este razonamiento está relacionado con la generación de hipótesis y creación de teorías. La abducción requiere de mucha creatividad e instinto para liberarse de la atadura del pensamiento racional, del conocimiento adquirido.

  Premisas:

  p: Los animales que vuelan tienen alas.

  r: Las palomas tienen alas.

  Conclusión:

  q: Las palomas vuelan.

  En la abducción la conclusión no es segura sino tan solo probable. Efectivamente en el ejemplo mostrado la conclusión se cumple para las palomas pero existen animales como

© Todos los derechos reservados

los pingüinos que tienen alas y no vuelan. Para determinar la validez del razonamiento se requieren otros métodos y técnicas que, en dependencia del contexto, permitirían asegurarla.

En cuanto a la validez de la conclusión obtenida en las diferentes formas de razonamiento, se puede concluir que mientras la deducción puede comprobarse empíricamente, en el caso de la inducción y la abducción, aunque no pueden ser válidas sin una confirmación empírica, estas confirmaciones no eliminan la posibilidad de existencia de una excepción. En este capítulo se estudiará el método deductivo, específicamente el deductivo proposicional.

4.2.1. Esquema de las estructuras deductivas

Todas las premisas son consideradas ciertas en un razonamiento deductivo, pues precisamente bajo el supuesto de que sean verdaderas y a partir de un conjunto de reglas lógicamente válidas, es que se pretende deducir la veracidad de las conclusiones. La conexión entre las premisas y las conclusiones se denomina estructura deductiva.

Las estructuras deductivas tienen el esquema siguiente:

$$p_1, p_2, \ldots, p_n \vdash q_1, q_2, \ldots, q_m$$

donde el conjunto de preposiciones P= $\{p_1, p_2, \ldots, p_n\}$ representa las premisas y el conjunto Q = $\{q_1, q_2, \ldots, q_m\}$, m > 0, las conclusiones, mientras que el signo "$\vdash$" es el signo de deducción o consecuencia lógica. La estructura representada se interpreta como "a partir de $p_1, p_2, \ldots, p_n$ se deduce $q_1, q_2, \ldots, q_m$".

**Nota*: También se puede usar el símbolo "$\Rightarrow$" en vez de "$\vdash$" para representar la deducción.

Ejemplo 4.2.1. Estructuras deductivas

Exprese en el lenguaje proposicional las siguientes estructuras deductivas:

a) El triángulo ABC es equilátero si tiene sus tres lados iguales. El triángulo ABC tiene tres lados iguales, entonces es equilátero.

b) Si llueve no vamos al cine, no llueve entonces deduzco que vamos al cine.

c) El paciente presenta problemas para orinar y tiene dolores suprapúbicos intensos, por tanto el paciente tiene cistitis.

d) El Equipo A es campeón si y sólo si clasifica y vence al Equipo D. Como el Equipo A es campeón, se deduce que el Equipo A vence al Equipo D.

Solución:

a) La conclusión de la estructura deductiva evidentemente es la proposición q: "el triángulo ABC es equilátero". La premisa sería la proposición p_1: "el triángulo ABC tiene tres lados iguales". Sin embargo únicamente con p_1 no se puede deducir q, es

© Todos los derechos reservados

necesaria otra premisa p_2 que defina que "el triángulo ABC es equilátero si tiene sus tres lados iguales". La estructura deductiva sería:

$p_1, p_2 \vdash q$

Sin embargo la expresión representada por p_2 encierra la condicional $p_1 \rightarrow q$, quedando la estructura deductiva como:

$p_1, p_1 \rightarrow q \vdash q$

b) En esta expresión se pueden identificar dos premisas, p_1: "si llueve no vamos al cine" y p_2: "llueve". Por otra parte se tiene una sola conclusión q_1: "vamos al cine". Luego con estas preposiciones se forma la siguiente estructura deductiva:

$p_1, p_2 \vdash q_1$

Un análisis más profundo de la proposición p_1 permite identificar la condicional $p_2 \rightarrow q_1$, por lo que la estructura deductiva queda como:

$p_2 \rightarrow q_1, p_2 \vdash q_1$

c) Las premisas en este razonamiento son:

p_1: El paciente presenta problemas para orinar.

p_2: El paciente tiene dolores suprapúbicos intensos.

Lo que conlleva a la conclusión q_1: el paciente tiene cistitis. Luego la estructura deductiva es la siguiente:

$p_1, p_2 \vdash q_1$

d) En la conformación de la estructura deductiva se utilizarán las proposiciones siguientes:

p_1: El Equipo A es campeón.

p_2: El Equipo A clasifica para la final.

q_1: El Equipo A vence al Equipo D.

Luego la premisa "El Equipo A es campeón si y sólo si clasifica y vence al Equipo D" se expresa como $p_1 \leftrightarrow (p_2 \wedge q_1)$. Luego la estructura deductiva completa sería:

$p_1 \leftrightarrow (p_2 \wedge q_1), p_1 \vdash q_1$

4.2.2. Validez de las deducciones

Una estructura deductiva es correcta si las premisas implican lógicamente la conclusión, es decir si todas las conclusiones son verdaderas en cada interpretación que haga ciertas todas las premisas, lo que se puede verificar usando las tablas de verdad. La tabla de verdad estaría formada por las posibles combinaciones de todas las variables involucradas en la estructura deductiva; una columna para los valores de verdad de cada una de las premisas y finalmente la conclusión o conclusiones. El ejemplo 4.2.2 muestra este proceso de validación.

© Todos los derechos reservados

EJEMPLO 4.2.2. VALIDEZ DE LAS ESTRUCTURAS DEDUCTIVAS

Determine la veracidad de las estructuras deductivas obtenidas en el ejemplo 4.2.1.

a) $p_1, p_1 \rightarrow q \vdash q$

Como se explicó anteriormente se debe realizar una tabla de verdad con las variables, las premisas y las conclusiones. La tabla 4.1 es la tabla correspondiente a este inciso.

Tabla 4.1: Tabla de verdad de la estructura deductiva "$p_1, p_1 \rightarrow q \vdash q$".

Variables		Premisas		Conclusión
p1	**q**	**p_1**	**$p_1 \rightarrow q$**	**q**
0	0	0	1	0
0	1	0	1	1
1	0	1	0	0
1	1	**1**	**1**	**1**

Luego se puede apreciar que para la única interpretación que hace verdadera ambas premisas también se cumple la conclusión q. Por tanto el razonamiento reflejado en la estructura deductiva "$p_1, p_1 \rightarrow q \vdash q$" es válido.

b) $p_2 \rightarrow q_1, p_2 \vdash q_1$

La tabla 4.2 muestra los valores de verdad de esta estructura deductiva.

Tabla 4.2: Tabla de verdad de la estructura deductiva "$p_2 \rightarrow q_1, p_2 \vdash q_1$".

Variables				Premisas		Conclusión
p_2	**q_1**	**$\overline{p}_2$**	**$\overline{q}1$**	**$p_2 \rightarrow \overline{q}_1$**	**$\overline{p}_2$**	**q_1**
0	0	1	1	**1**	**1**	**0**
0	1	1	0	**1**	**1**	**1**
1	0	0	1	1	0	0
1	1	0	0	0	0	1

En la tabla se señalan dos interpretaciones para las cuales se cumplen todas las premisas, sin embargo en una de ellas, señalada en rojo, la conclusión no es verdadera, de manera que el razonamiento representado es inválido.

c) $p_1, p_2 \vdash q_1$

Como las premisas coinciden con las variables se obviarán las columnas correspondientes a las variables, quedando la tabla de verdad siguiente:

© TODOS LOS DERECHOS RESERVADOS

Tabla 4.3: Tabla de verdad de la estructura deductiva "$p_1, p_2 \vdash q_1$".

Premisas		Conclusión
p1	**p2**	**q1**
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	**1**	**0**
1	**1**	**1**

Al igual que en el inciso anterior, existe al menos una interpretación para la cual se cumplen las premisas y la conclusión no. Dicha interpretación está señalada en rojo y es la responsable de que el razonamiento representado en el inciso c sea inválido.

d) $p_1 \leftrightarrow (p_2 \wedge q_1), p_1 \vdash q_1$

La tabla de verdad correspondiente se muestra a continuación:

Tabla 4.4: Tabla de verdad de la estructura deductiva "$p_1 \leftrightarrow (p_2 \wedge q_1), p_1 \vdash q_1$".

Variables			Premisas		Conclusión
p1	**p2**	**q1**	**p1 ↔ (p2 ∧ q1)**	**p1**	**q1**
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	0	1	0
1	1	1	**1**	**1**	**1**

Según se aprecia en la última fila de la tabla 4.3, la única interpretación que hace verdaderas ambas premisas también tiene una conclusión con valor 1, por lo que la deducción realizada es correcta.

Las estructuras deductivas que no son válidas se denominan falacias. A continuación se ilustran algunas falacias comunes.

© Todos los derechos reservados

Falacia "*Negación del antecedente*"

Algunas falacias surgen de la negación del antecedente, por ejemplo:

Si Claudia tiene las joyas robadas en su bolso, entonces es culpable.

Las joyas no están en el bolso de Claudia.

Por tanto, Claudia es inocente.

Para analizar la validez de este razonamiento se declaran las proposiciones siguientes:

p: Claudia tiene las joyas robadas en su bolso.

q: Claudia es culpable.

Luego la estructura deductiva y su tabla de verdad correspondiente serían:

$$p \rightarrow q,\ p \vdash q$$

Tabla 4.5: Tabla de verdad de la estructura deductiva "$p \rightarrow q,\ p \vdash q$".

Variables		Premisas		Conclusión
p	**q**	**p → q**	**p**	**q**
0	0	**1**	**1**	**1**
0	1	**1**	**1**	**0**
1	0	0	0	1
1	1	1	0	0

Note que no siempre que ambas premisas son verdaderas la conclusión también lo es, la primera interpretación (fila 1) hace verdaderas ambas premisas y la conclusión; sin embargo en la segunda interpretación (fila 2) la conclusión no se cumple mientras que las premisas sí son verdaderas. Por tanto se concluye que el razonamiento es inválido. Efectivamente, el hecho de que Claudia no lleve las joyas robadas en el bolso no significa que no sea culpable, ya que si ella las robó bien pudo haberla escondido en algún sitio.

Falacia "*Afirmación de la conclusión*"

Otra forma de falacia es afirmar la conclusión, por ejemplo:

Cuando llueve el jardín está mojado.

El jardín está mojado.

Entonces llueve.

© Todos los derechos reservados

Las proposiciones que se utilizarán en la validación son las siguientes:

p: Llueve.

q: el jardín está mojado.

A continuación se muestra la estructura deductiva y la tabla de verdad correspondiente:

$$p \rightarrow q, q \vdash p$$

Tabla 4.6: Tabla de verdad de la estructura deductiva "$p \rightarrow q, q \vdash p$".

Variables		Premisas		Conclusión
p	**q**	**p → q**	**q**	**p**
0	0	1	0	0
0	1	**1**	**1**	**0**
1	0	0	0	1
1	1	**1**	**1**	**1**

Igual que en la falacia analizada anteriormente, no para todas las interpretaciones que hacen ciertas ambas premisas se cumple también la conclusión. En la primera interpretación (fila 2) la conclusión no se cumple mientras que las premisas sí; y en la cuarta interpretación (fila 4) son verdaderas ambas premisas y la conclusión. Luego el razonamiento es inválido. Analizando bien el razonamiento se puede ver que el hecho de que el jardín esté mojado no significa necesariamente que haya llovido, por ejemplo puede estar mojado porque alguien lo hayan regado.

Las estructuras deductivas no precisan obligatoriamente de la existencia de premisas pero sí debe obtenerse al menos una conclusión. En este caso para confirmar la veracidad de la deducción, las conclusiones deben ser ciertas para todas las interpretaciones en la tabla de verdad, es decir, deben ser tautologías. El ejemplo 4.2.3 es una muestra de este tipo particular de estructuras deductivas.

Ejemplo 4.2.3.

Estructuras deductivas sin premisas.

Sea p: "La puerta está abierta" una deducción natural es "la puerta está abierta o no está abierta".

$$\vdash p \vee \overline{p}$$

© Todos los derechos reservados

Tabla 4.7: Tabla de verdad de la estructura deductiva correspondiente al ejemplo 4.2.3.

Variable	Conclusión
p	$\mathbf{p \vee \overline{p}}$
0	1
1	1

Como se puede apreciar en la tabla de verdad (tabla 4.7) la conclusión es una tautología por lo que se considera éste un razonamiento válido.

4.3. REGLAS DE INFERENCIA

En ocasiones resulta engorroso o no viable comprobar la veracidad de una deducción a partir de su tabla de verdad, ya que pudiera ser excesivamente larga o pudiera necesitarse determinar la veracidad de fórmulas en las que no intervienen variables proposicionales a las cuales asignarles valores de verdad. En estos casos se utiliza otro método basado en reglas de inferencias.

Las reglas de inferencias son un conjunto de reglas válidas que permiten demostrar la veracidad de una estructura deductiva partiendo de un conjunto de premisas (asumidas como ciertas) hasta deducir las conclusiones deseadas. Las reglas de inferencia tienen su origen en las implicaciones lógicas. Su estructura está formada por un conjunto premisas (P_i) separadas por una línea horizontal de la conclusión (Q) de la regla

:

$$\begin{array}{c} P_1 \\ P_2 \\ \vdots \\ P_n \\ \hline \therefore Q \end{array}$$

El símbolo $\therefore$ se lee “por lo tanto”. La tabla 4.8 muestra un resumen de las principales reglas de inferencia.

© Todos los derechos reservados

Tabla 4.8: Reglas de inferencia.

Nombre	Regla	Nombre	Regla
Modus Ponens	P $P \rightarrow Q$ $\therefore Q$	Introducción de la disyunción o Adición	P $\therefore P \vee Q$
Modus Tollens	$P \rightarrow Q$ Q $\therefore \overline{P}$	Simplificación	$P \wedge Q$ $\therefore P$
Silogismo Hipotético	$P \rightarrow Q$ $Q \rightarrow R$ $\therefore P \rightarrow R$	Prueba por casos	$P \rightarrow R$ $Q \rightarrow R$ $\therefore (P \vee Q) \rightarrow R$
Silogismo Disyuntivo	$P \vee Q$ P $\therefore Q$	Dilema constructivo	$P \rightarrow Q$ $R \rightarrow S$ $P \vee R$ $\therefore Q \vee S$
Introducción de la conjunción	P Q $\therefore P \wedge Q$	Dilema destructivo	$P \rightarrow Q$ $R \rightarrow S$ $\overline{Q} \vee \overline{S}$ $\therefore (\overline{P} \vee \overline{R})$

El proceso de validación de estructuras deductivas, utilizando reglas de inferencia, se compone de un conjunto de pasos numerados que contienen, cada uno, una estructura deductiva. Cada paso será justificado a partir del anterior apoyándose en una regla de inferencia. La justificación se mostrará a un lateral del paso realizado. La última estructura deductiva de la demostración debe tener como conclusión la proposición que se desea verificar y, basándose en la validez de las transiciones entre los pasos, se puede asegurar que la misma es correcta. El esquema explicado se representa de la siguiente forma:

(# del paso) Conclusión < Justificación >

Ejemplo 4.3.1. Reglas de inferencia

Identifique la regla de inferencia utilizada en cada paso. Cada letra mayúscula representa una fórmula bien formada.

a) 1. $A \rightarrow B$

2. $(A \rightarrow C) \vee (D \vee \overline{B})$

3. $\overline{D}$

4. $\overline{A \rightarrow C}$

$\therefore A \wedge (A \rightarrow B)$

© Todos los derechos reservados

5. $(D \vee \overline{B})$
6. $\overline{B}$
7. A
8. $A \wedge (A \rightarrow B)$

b)
1. $P \rightarrow \overline{Q}$
2. $Q \wedge R$

$\therefore \overline{P} \vee S$

3. $(Q \rightarrow \overline{P})$
4. Q
5. $\overline{P}$
6. $\overline{P} \vee S$

c)
1. $(A \wedge B) \rightarrow C$
2. $\overline{A} \rightarrow D$
3. C
4. $\overline{B} \rightarrow E$
5. $F \rightarrow (\overline{D} \wedge \overline{E})$

$\therefore \overline{F}$

6. $(\overline{A \wedge B})$
7. $\overline{A} \vee \overline{B}$
8. $D \vee E$
9. $(\overline{D \wedge E}) \rightarrow \overline{F}$
10. $(D \vee E) \rightarrow \overline{F}$
11. F

Solución:

a)
1. $A \rightarrow B$
2. $(A \rightarrow C) \vee (D \vee \overline{B})$
3. $\overline{D}$
4. $\overline{A \rightarrow C}$

$\therefore A \wedge (A \rightarrow B)$

5. $(D \vee \overline{B})$ — Silogismo Disyuntivo con las premisas 2 y 4.
6. $\overline{B}$ — Silogismo Disyuntivo con las premisas 5 y 3.

© Todos los derechos reservados

7.	A	Modus Tollens con las premisas 1 y 6.
8.	$A \wedge (A \rightarrow B)$	Introducción de la conjunción con las premisas 1 y 7.

b)

1. $P \rightarrow \overline{Q}$
2. $Q \wedge R$

$\therefore \overline{P} \vee S$

3.	$Q \rightarrow \overline{P}$	Contrarrecíproco a la premisa 1.
4.	Q	Simplificación con la premisa 2.
5.	$\overline{P}$	Modus Ponens con las premisas 3 y 4.
6.	$\overline{P} \vee S$	Introducción de la disyunción con la premisa 5.

c)

1. $(A \wedge B) \rightarrow C$
2. $\overline{A} \rightarrow D$
3. C
4. $\overline{B} \rightarrow E$
5. $F \rightarrow (\overline{D} \wedge \overline{E})$

$\therefore \overline{F}$

6.	$(A \wedge B)$	Modus Tollens con las premisas 1 y 3.
7.	$\overline{A} \vee \overline{B}$	Aplicando las Leyes de Morgan a la premisa 6.
8.	$D \vee E$	Dilema constructivo con las premisas 2, 4 y 7.
9.	$(\overline{D} \wedge \overline{E}) \rightarrow \overline{F}$	Aplicando el contrarrecíproco a la premisa 5.
10.	$(D \vee E) \rightarrow \overline{F}$	Aplicando las Leyes de Morgan a la premisa 9.
11.	F	Modus Ponens con las premisas 8 y 10.

EJEMPLO 4.3.2.

Reglas de inferencia.

Utilizando las reglas de inferencia demuestre la validez de los razonamientos siguientes, donde cada letra mayúscula representa una fórmula bien formada. Indique en la justificación la regla de inferencia utilizada en cada paso.

© TODOS LOS DERECHOS RESERVADOS

a) 1. $A \vee B$

2. $B \rightarrow C$

3. $C \rightarrow D$

4. $\overline{A}$

$\therefore D$

b) 1. $P \rightarrow Q$

2. $P \vee (Q \vee \overline{R})$

3. $\overline{Q}$

$\therefore \overline{R} \wedge \overline{Q}$

c) 1. $(H \rightarrow I) \rightarrow (J \rightarrow K)$

2. $(H \rightarrow I) \vee \overline{K}$

3. $N \rightarrow (L \rightarrow M)$

4. $K \vee \overline{(L \rightarrow M)}$

5. $\overline{(J \rightarrow K)}$

$\therefore \overline{N}$

© Todos los derechos reservados

Solución:

a) 1. $A \vee B$

2. $B \rightarrow C$

3. $C \rightarrow D$

4. $\overline{A}$

$\therefore D$

5. B	Silogismo Disyuntivo con las premisas 1 y 4.
6. $B \rightarrow D$	Silogismo Hipotético con las premisas 2 y 3.
7. D	Modus Ponens con las premisas 5 y 6.

b) 1. $P \rightarrow Q$

2. $P \vee (Q \vee \overline{R})$

3. $\overline{Q}$

$\therefore \overline{R} \wedge \overline{Q}$

4.	$\overline{P}$	Modus Tollens con las premisas 1 y 3.
5.	$Q \vee \overline{R}$	Silogismo Disyuntivo con las premisas 2 y 4.
6.	$\overline{R}$	Silogismo Disyuntivo con las premisas 3 y 5.
7.	$\overline{R} \wedge \overline{Q}$	Introducción de la conjunción con la premisa 3 y 6.

c)

1.	$(H \rightarrow I) \rightarrow (J \rightarrow K)$	
2.	$(H \rightarrow I) \vee \overline{K}$	
3.	$N \rightarrow (L \rightarrow M)$	
4.	$K \vee \overline{(L \rightarrow M)}$	
5.	$\overline{(J \rightarrow K)}$	
	$\therefore \overline{N}$	
6.	$\overline{(H \rightarrow I)}$	Modus Tollens con las premisas 1 y 5.
7.	$\overline{K}$	Silogismo Disyuntivo con las premisas 2 y 6.
8.	$\overline{(L \rightarrow M)}$	Silogismo Disyuntivo con las premisas 4 y 7.
9.	$\overline{N}$	Modus Tollens con las premisas 3 y 8.

4.4. DEMOSTRACIONES

El razonamiento deductivo es ampliamente utilizado en las *demostraciones* de proposiciones matemáticas, cuya veracidad se establece partiendo de la veracidad de un conjunto de proposiciones, algunas aceptadas como verdaderas, otras generalmente demostradas con anterioridad. El conjunto de las proposiciones aceptadas como verdaderas sin demostración previa, se denominan *axiomas o postulados* y constituyen el punto de partida para las demostraciones. Las proposiciones cuya veracidad ha sido demostrada previamente se denominan teoremas. En este manual se presentan tres técnicas de demostración: directa, por contradicción y por inducción matemática.

4.4.1. DEMOSTRACIÓN DIRECTA

Gran parte de los teoremas matemáticos están compuestos por proposiciones condicionales del tipo:

$$(p_1 \wedge p_2 \wedge \ldots \wedge p_n) \rightarrow q$$

Este tipo de proposiciones únicamente son falsas cuando $(p_1 \wedge p_2 \wedge \ldots \wedge p_n)$ es verdadera y la conclusión q es falsa; recuerde que en la tabla de verdad de la implicación si el antecedente es

© TODOS LOS DERECHOS RESERVADOS

falso la proposición es verdadera independientemente del valor de verdad de la conclusión. Por tanto para llevar a cabo una demostración formal basada en el método directo se parte de reconocer a p_1, p_2,... y p_n como axiomas (verdaderas), luego, para que el razonamiento (proposición condicional) sea válido se debe demostrar que q es verdadera utilizando otros axiomas o teoremas.

EJEMPLO 4.4.1. DEMOSTRACIÓN DIRECTA

Utilice el método de demostración directa para determinar la validez de los argumentos siguientes:

a) Si ***x*** y ***y*** son números enteros pares, entonces ***13x + y*** es un número entero par.

Solución:

a) Si ***x*** y ***y*** son números enteros pares, entonces ***13x + y*** es un número entero par.

Siguiendo el método de demostración directa se parte de que la hipótesis "***x*** y ***y*** son números enteros pares" es verdadera y se intenta demostrar que la conclusión "***13x + y*** es un número entero par", también lo es.

Los números pares cumplen con la forma 2n, por tanto:

$x = 2n_1 \qquad y = 2n_2$

$13\,(2n_1) + (2n_2) = 2\,(13n_1 + n_2)$

$= 2n$, donde $n = 13n_1 + n_2$

Al obtenerse como resultado un número que cumple con la forma 2n se puede afirmar que si ***x*** y ***y*** son número enteros pares, entonces "13x + y" da como resultado un número entero par. Por lo que queda demostrada la validez del razonamiento.

4.4.2. DEMOSTRACIÓN POR CONTRADICCIÓN O REDUCCIÓN AL ABSURDO

La demostración por contradicción, también conocida como demostración por reducción al absurdo, constituye una demostración indirecta. El procedimiento de la demostración por contradicción es semejante al utilizado en el método directo, con la diferencia de que se incluye entre los axiomas la negación de la conclusión. El objetivo de la demostración es llegar a una contradicción del tipo $r \wedge r$, donde r representa cualquier proposición ya sea de las premisas o de las obtenidas en algún paso de la demostración.

© TODOS LOS DERECHOS RESERVADOS

EJEMPLO 4.4.2. DEMOSTRACIÓN POR CONTRADICCIÓN

Demuestre la validez de los argumentos siguientes utilizando el método de demostración por contradicción.

a)
1. $A \rightarrow B$
2. $C \rightarrow D$
3. $\overline{B} \vee \overline{D}$
4. $\overline{A}$
5. $(E \wedge F) \rightarrow C$

$\therefore\ (\overline{E \wedge F})$

b) Si ***5a + 2b*** es un número entero impar entonces **a** es un número entero impar.

Solución:

a) Recuérdese que el procedimiento se basa en probar que el incumplimiento de la proposición (conclusión falsa) conduce a una contradicción lógica. Por tanto el primer paso es incorporar entre los axiomas (premisas) la negación de la conclusión.

1. $A \rightarrow B$
2. $C \rightarrow D$
3. $\overline{B} \vee \overline{D}$
4. $\overline{A}$
5. $(E \wedge F) \rightarrow C$
6. $(E \wedge F)$

$\therefore\ (\overline{E \wedge F})$

Luego se aplican las reglas de inferencia y las leyes de la lógica en busca de una contradicción del tipo $r \wedge \overline{r}$.

7.	$\overline{A} \vee \overline{C}$	*Dilema destructivo con la premisa 1, 2 y 3.*
8.	A	*Ley de la doble negación a la premisa 4.*
9.	$\overline{C}$	*Silogismo Disyuntivo con las premisas 7 y 8.*
10.	C	*Modus Ponens con las premisas 5 y 6.*
11.	$\overline{C} \wedge C$	*Introducción de la conjunción con las premisas 9 y 10.*

© TODOS LOS DERECHOS RESERVADOS

Como se puede apreciar en el paso 11 se ha obtenido una contradicción, por tanto el razonamiento es válido.

b) Si ***5a + 2b*** es un número entero impar entonces ***a*** es un número entero impar.

La proposición tiene la forma $p \rightarrow q$.

p: ***5a + 2b*** es un número entero impar.

q: ***a*** es un número entero impar.

El primer paso es negación de la conclusión utilizándola como axioma y tratar de encontrar una contradicción.

q: ***a*** es un número entero par.

**Nota*: Los números pares tienen la forma "2n" y los impares "2n+1".

5a + 2b

Como ***a*** representa un número par:

5(2t) + 2b

Luego sacando como factor común el número 2:

2(5t + b) = 2n, donde n = 5t + b

Como se puede apreciar independientemente del valor de n, se obtendría como resultado un número par ya que cumple con la forma 2n. En otras palabras se obtuvo que "5a + 2b es un número entero par" lo que contradice la premisa dada en el razonamiento ($p \wedge \overline{p}$). Por lo que queda demostrada la validez del razonamiento.

© Todos los derechos reservados

4.4.3. Demostración por inducción matemática

Este tipo de demostración se utiliza cuando los enunciados tienen una proposición abierta en una variable ***n***, y es necesario demostrar que tal proposición se cumple para todos los elementos ***n*** que pertenecen a un subconjunto infinito dado sobre los números enteros, por ejemplo:

Para todo número natural ***n***, se cumple que: $1 + 3 + 5 + ... + (2n - 1) = n^2$.

**Nota:* Este enunciado será demostrado posteriormente en el ejemplo 4.4.3.

Principio de inducción matemática

Sea *p(n)* una afirmación, verdadera o falsa, donde n representa cada número entero positivo; si $p(n_0)$ es verdadera y se cumple que si $p(n_i)$ es verdadera entonces $p(n_i+1)$ es verdadera; se puede concluir que *p(n)* es verdadera para todo ***n*** entero positivo.

La condición "$p(n_0)$ es verdadera" constituye el paso base, donde se demuestra la validez de la proposición "*p*" evaluada en el caso base "n_0" que constituye la condición inicial del razonamiento que se está demostrando. La condición "si $p(n_i)$ es verdadera entonces $p(n_i+1)$ es verdadera" representa el paso *inductivo*.

Por tanto el primer paso de la demostración por inducción matemática es verificar la validez de $p(n_0)$ y luego, asumiendo que $p(n_i)$ es verdadera, se demuestra que $p(n_i+1)$ también es verdadera.

EJEMPLO 4.4.3.

Demostración por inducción matemática.

Demuestre la validez de los argumentos siguientes utilizando el método de demostración por inducción matemática.

a) $1 + 3 + 5 + \ldots + (2n - 1) = n^2$, para todo número natural n.

b) Cada número de Fibonacci es el promedio del término que se encuentra dos posiciones antes y el término que se encuentra una posición después. Es decir:

$$f_{n-1} + f_{n-2} = \frac{f_{n-2} + f_{n-1}}{2}$$
$$f_0 = 0$$
$$f_1 = 1$$

Para n=2, 3, 4...

c) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, donde ***n*** es un número entero positivo.

Solución:

a) $1 + 3 + 5 + \ldots + (2n - 1) = n^2$

La demostración se realiza sobre la base de la proposición
$p(n) = 1 + 3 + 5 + \ldots + (2n - 1) = n^2$

Paso base: n = 1

$p(1) = 2(1) - 1 = 1^2$

$1 = 1$

Paso inductivo: $n = n_i$

- Se asume que $p(n_i)$ es verdadera, es decir:

  $p(n_i) = 1 + 3 + 5 \ldots + (2n_i - 1) = n_i^2$

- Se comprueba la validez de $p(n_i+1)$:

  $p(n_i+1) = \underbrace{1 + 3 + 5 \ldots + (2n_i - 1)}_{p(n_i)} + (2[n_i + 1] - 1) = (n_i+1)^2$

  $p(n_i+1) = p(n_i) + (2n_i + 2 - 1) = (n_i+1)^2$

  $p(n_i+1) = n_i^2 + 2n_i + 1 = (n_i+1)^2$

  $p(n_i+1) = n_i^2 + 2n_i + 1 = n_i^2 + 2n_i + 1$

© TODOS LOS DERECHOS RESERVADOS

Por tanto queda demostrado que la proposición
$1 + 3 + 5 + \ldots + (2n - 1) = n^2$ es verdadera.

b) $$f_{n-1} + f_{n-2} = \frac{f_{n-2} + f_{n+1}}{2}$$

Paso base: n=2

$$f_1 + f_0 = \frac{f_0 + f_3}{2}$$

$$1 + 0 = \frac{0+2}{2}$$

$$1 = 1$$

Paso inductivo: $n = n_i$

- Se asume que $p(n_i)$ es verdadera, es decir:

$$f_{n_i} = f_{n_i-1} + f_{n_i-2} = \frac{f_{n_i-2} + f_{n_i+1}}{2}$$

- Se comprueba la validez de $p(n_i+1)$:

$$f_{n_i+1} = f_{(n_i+1)-1} + f_{(n_i+1)-2} = \frac{f_{(n_i+1)-2} + f_{(n_i+1)-1}}{2}$$

$$f_{n_i+1} = f_{n_i} + f_{n_i-1} = \frac{f_{n_i-1} + f_{n_i+2}}{2}$$

$$f_{n_i+1} = \overbrace{\frac{f_{n_i-2} + f_{n_i+1}}{2}} + f_{n_i-1} = \frac{f_{n_i-1} + f_{n_i+2}}{2}$$

$$f_{n_i+1} = \frac{f_{n_i-2} + f_{n_i+1} + 2f_{n_i-1}}{2} = \frac{f_{n_i-1} + f_{n_i+2}}{2}$$

$$f_{n_i+1} = \frac{(f_{n_i-2} + f_{n_i-1}) + f_{n_i+1} + f_{n_i-1}}{2} = \frac{f_{n_i-1} + f_{n_i+2}}{2}$$

Note que $f_{n_i-2} + f_{n_i-1} = f_{n_i}$ por tanto:

$$f_{n_i+1} = \frac{f_{n_i} + f_{n_i+1} + f_{n_i-1}}{2} = \frac{f_{n_i-1} + f_{n_i+2}}{2}$$

Igualmente se puede apreciar que $f_{n_i} + f_{n_i+1} = f_{n_i+2}$, sustituyendo se obtiene que:

$$f_{n_i+1} = \frac{f_{n_i+2} + f_{n_i-1}}{2} = \frac{f_{n_i-1} + f_{n_i+2}}{2}$$

Así queda demostrada la validez de la proposición $f_{n-1} + f_{n-2} = \frac{f_{n-2} + f_{n+1}}{2}$.

© Todos los derechos reservados

c) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, donde ***n*** es un número entero positivo.

Paso base: n=1

$$\sum_{i=1}^{1} i = \frac{1(1+1)}{2}$$

$$1 = \frac{2}{2}$$

1 = 1

Paso inductivo: n= n_i

- Se asume que $p(n_i)$ es verdadera, es decir:

$$\sum_{i=1}^{n_i} i = \frac{n_i(n_i+1)}{2}$$

$$\sum_{i=1}^{n_i} i = 1+2+3+\cdots+n_i$$

$$1+2+3+\cdots+n_i = \frac{n_i(n_i+1)}{2}$$

- Se comprueba la validez de $p(n_i+1)$:

$$\sum_{i=1}^{n_i+1} i = \frac{(n_i+1)(n_i+1+1)}{2}$$

$$\sum_{i=1}^{n_i+1} i = 1+2+3+\cdots+n_i+(n_i+1)$$

$$\underbrace{1+2+3+\cdots+n_i}+(n_i+1) = \frac{(n_i+1)(n_i+1+1)}{2}$$

$$\sum_{i=1}^{n_i} i + (n_i+1) = \frac{(n_i+1)(n_i+2)}{2}$$

$\frac{n_i(n_i+1)}{2} + (n_i+1) = \frac{(n_i+1)(n_i+2)}{2}$, sacando factor común (n_i+1)

$$(n_i+1)(\frac{n_i}{2}+1) = \frac{(n_i+1)(n_i+2)}{2}$$

$$(n_i+1)(\frac{n_i+2}{2}) = \frac{(n_i+1)(n_i+2)}{2}$$

$$\frac{(n_i+1)(n_i+2)}{2} = \frac{(n_i+1)(n_i+2)}{2}$$

De esta forma se demuestra que la proposición $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ es verdadera.

© Todos los derechos reservados

4.5. VERIFICACIÓN FORMAL DE PROGRAMAS A TRAVÉS DE LA LÓGICA DEDUCTIVA

Supongamos que hemos diseñado un algoritmo para solucionar un problema y hemos escrito un programa para implementarlo. ¿Cómo podemos estar seguros de que este programa produce siempre la respuesta correcta? Tras haber depurado todos los errores de sintaxis, podemos probar el programa con un muestreo de entradas. No será correcto si produce un resultado incorrecto para alguna de las entradas probadas. Pero aun en el caso de que el programa ofrezca una respuesta correcta para todas las entradas muestreadas, puede que no siempre produzca la respuesta correcta (a no ser que se comprueben todas las entradas posibles). Necesitamos una demostración para asegurar que el programa siempre dará la salida correcta.

La verificación de programas, en otras palabras la demostración de que los programas son correctos, involucra reglas de inferencia y técnicas de demostración descritas en este capítulo, incluyendo la inducción matemática muestra de que los temas tratados en este capítulo son de gran importancia para las ciencias de la computación. Como un programa incorrecto puede conducir a resultados desastrosos, se ha desarrollado una vasta metodología para su verificación. Se ha dedicado un gran esfuerzo a automatizar la verificación de programas con la intención de que se pueda realizar en ordenador. No obstante, sólo se han conseguido progresos limitados. De hecho, algunos matemáticos y teóricos de la informática argumentan que nunca se podrá mecanizar de forma realista la demostración de que programas complejos son correctos.

En esta sección se presentarán algunos de los conceptos y métodos utilizados para demostrar que los programas son correctos. No obstante, en este manual no se desarrollará una metodología completa de la verificación de programas. Esta sección se ha pensado como una breve introducción al área de la verificación de programas que enlaza reglas de lógica, técnicas de demostración y el concepto de algoritmo.

© Todos los derechos reservados

4.5.1. Verificación formal de algoritmos mediante axiomas de la lógica de Hoare

La lógica de Hoare da reglas para probar la corrección de las instrucciones básicas de un lenguaje de programación (asignación, composición secuencial, condicional, composición iterativa...). Estas reglas permiten calcular de manera mecánica precondiciones válidas a partir de una postcondición dada para asignaciones, composiciones secuenciales y sentencias condicionales.

Se dice que un programa es correcto si produce la salida correcta para toda entrada de datos posible. Una demostración de que un programa es correcto consta de dos partes. La primera demuestra que se obtiene la respuesta correcta si el programa termina. Esta parte de la demostración establece la ***verificación parcial*** del programa. La segunda parte de la demostración demuestra que el programa siempre termina.

Para especificar qué entendemos con que un programa produzca la respuesta correcta, utilizamos dos proposiciones. La primera es la ***afirmación inicial***, que da las propiedades que debe cumplir la entrada. La segunda es la ***afirmación final***, que da las propiedades que debería cumplir la salida si

el programa hiciese lo que ideamos. Para comprobar un programa, debemos proporcionar afirmaciones iniciales y finales.

Definición 4.5.1: Un programa, o segmento[1] de un programa, *S* se dice que es parcialmente correcto respecto a la afirmación inicial *q* si siempre que *p* es verdadera para los valores de entrada de *S* y *S* termina, entonces *q* es verdadera para los valores de salida de *S*. La notación *p{S}q* indica que el programa, o segmente de programa, *S* es parcialmente correcto respecto a las afirmaciones inicial *p* y final *q*.

Nota: La notación *p{S}q* se conoce como la terna de Hoare, por Tony Hoare, quien introdujo el concepto de verificación parcial.

Ten en cuenta que la noción de corrección parcial es independiente del hecho de que el programa termine o no; se basa en si el programa se ajusta o no a lo esperado en el caso de que éste termine.

El siguiente ejemplo ilustrará el concepto de afirmaciones iniciales y finales.

EJEMPLO 4.5.1.

Muestra que el segmento de programa:

```
y = 2;
z = x + y;
```

es correcto con respecto a la afirmación inicial *p*: $x = 1$, y la afirmación final *q*: $z = 3$.

Solución:

Supongamos que *p* es verdadera, por lo que el programa comienza con el valor de *x* igual a 1. Entonces a *y* se le asigna el valor 2 y a *z* la suma de *x* e *y*, que es 3. Por tanto, *S* es correcto con respecto a la afirmación inicial *p* y la final *q*. Así, *p{S}q* es verdadera.

Esta misma estrategia es utilizada en la actualidad por los programas que realizan la función de calificación automática en concursos de programación competitiva, estos cuentan con archivos de entrada y salida según el problema a calificar.

4.5.2. REGLAS DE INFERENCIA PARA VERIFICAR PROGRAMAS

Una regla de inferencia útil demuestra que un programa es correcto dividiendo el programa en una serie de subprogramas y mostrando entonces que cada uno de ellos es correcto; es decir utiliza una estrategia de divide y vencerás.

1. Segmento de un programa se refiere a una parte sintácticamente consistente de un programa.

© TODOS LOS DERECHOS RESERVADOS

Supongamos que el programa S se divide en los subprogramas S_1 y S_2. Escribimos $S = S_1; S_2$ para indicar que S está formado por S_1 seguido de S_2. Supongamos que hemos establecido que S_1 es parcialmente correcto respecto de la afirmación inicial p y la final q y que S_2 es parcialmente correcto respecto de la afirmación inicial q y la final r. Se sigue que si p es verdadera y S_1 se ejecuta y termina, entonces q es verdadera, y si q es verdadera y S_2 se ejecuta y termina, entonces r es verdadera. Esta regla de inferencia, llamada ***regla de la composición***, se puede enunciar como:

$$\frac{\begin{array}{c} p\{S_1\}q \\ q\{S_2\}r \end{array}}{\therefore p\{S_1;\ S_2\}r}$$

Analizaremos algunas reglas de inferencia para segmentos de programas relacionados con sentencias condicionales y bucles. Como un programa puede dividirse en subprogramas o segmentos para comprobar si es correcto, estas reglas nos permitirán verificar muchos programas diferentes.

© Todos los derechos reservados

Sentencia condicional

Supongamos que un segmento de programa tiene la forma general:

if *condición* **then**

S

donde S es un bloque de instrucciones. Entonces S se ejecuta si *condición* es verdadera y no se ejecuta cuando *condición* es falsa. Para verificar que este segmento es correcto con respecto a la afirmación inicial p y la afirmación final q, debemos hacer dos cosas. Primero, se debe demostrar que cuando p es verdadera y *condición* también lo es, entonces q es verdadera al terminar S. Segundo, se debe ver que cuando p es verdadera y *condición* es falsa, entonces q es verdadera (puesto que en este caso S no se ejecuta).

Estas ideas conducen a la siguiente regla de inferencia:

$$\frac{\begin{array}{c} (p \wedge \text{condición})\{S\}q \\ (p \wedge \neg\text{condición}) \rightarrow q \end{array}}{\therefore p\{\textbf{if}\ \text{condición}\ \textbf{then}\ S\}q}$$

EJEMPLO 4.5.2.

Verifica que el segmento de programa es correcto con respecto a la afirmación inicial ***V*** y la afirmación final $y \geq x$.

Solución:

Cuando la afirmación inicial es verdadera y $x > y$, se lleva a cabo la asignación $y = x$. Por tanto, la afirmación final $y \geq x$ es verdadera en este caso. Además, cuando la afirmación inicial es verdadera y $x > y$ es falsa, es decir, $x < y$, la afirmación final es de nuevo verdadera. Por tanto, utilizando la regla de inferencia para segmentos de programas de este tipo, este programa es correcto con respecto a las afirmaciones iniciales y finales dadas.
Análogamente, supongamos que un programa tiene una sentencia de la forma:

if *condición* **then**
S_1
else
S_2

Si *condición* es verdadera, entonces se ejecuta S_1; si condición es falsa, entonces se ejecuta S_2. Para verificar que este segmento de programa es correcto con respecto a la afirmación inicial p y a la afirmación final q, se deben hacer dos cosas. Primero, debemos mostrar que cuando p es verdadera y condición es verdadera, entonces q es verdadera tras terminar S_1. Segundo, debemos mostrar que cuando p es verdadera y condición es falsa, entonces q es verdadera tras terminar S_2. Esto conduce a la siguiente regla de inferencia:

$$\frac{\begin{array}{c}(p \wedge \textit{condición})\{S_1\}q \\ (p \wedge \neg\textit{condición})\{S_2\}q\end{array}}{\therefore p\{\textbf{if}\ \text{condición}\ \textbf{then}\ S_1\ \text{else}\}q}$$

EJEMPLO 4.5.3.

Verifica que el segmento de programa es correcto con respecto a la afirmación inicial ***V*** y la afirmación final $abs = |x|$.

Solución:

Debemos demostrar dos cosas. Primero, debemos ver que si la afirmación inicial es verdadera y $x < 0$, entonces $abs = |x|$ Esto es correcto, puesto que cuando $x < 0$ la asignación de la sentencia $abs = |x|$ fija el valor de abs igual a $-x$, que es, por definición, $|x|$

© TODOS LOS DERECHOS RESERVADOS

para $x < 0$. Segundo, debemos ver que si la afirmación inicial es verdadera y $x < 0$ es falsa, por lo que $x \geq 0$, entonces $abs = -x$. Esto también es correcto, puesto que en este caso el programa utiliza la sentencia de asignación $abs = x$ y $x = |x|$ por definición para $x \geq 0$, por lo que $abs = x$. Así, usando la regla de inferencia para segmentos de programas de este tipo, este segmento es correcto con respecto a las afirmaciones inicial y final dadas.

Invariantes del bucle

Vamos a describir ahora demostraciones de que un bucle **while** es correcto. Para desarrollar una regla de inferencia para segmentos de programas del tipo:

while *condición*

S

Tener en cuenta que *S* se ejecuta repetidamente hasta que condición se hace falsa. Debemos elegir una afirmación que siga siendo verdadera cada vez que se ejecuta *S*. Tal afirmación se llama ***invariante del bucle***. En otras palabras, p es un invariante del bucle si $(p \wedge condición)\{S\}p$ es verdadera.

Supongamos que p es un invariante del bucle. Se sigue que si p es verdadera antes de ejecutar el segmento de programa, p y $\neg condición$ son verdaderas tras acabar, si acaba. Esta regla de inferencia es:

$$\frac{(p \wedge \neg condición)\{S\}q}{\therefore p\{\textbf{white}\ condición\ S\}\ (\neg condición \wedge p)}$$

EJEMPLO 4.5.4.

Necesitamos un invariante del bucle para verificar que el segmento de programa termina con *factorial=n!* cuando n es un entero positivo.

```
i = 1
factorial = 1
while i < n
begin
i = i + 1
factorial = factorial · i
end
```

Sea p la afirmación $factorial = i!$ e $i \leq n$. Debemos demostrar primero que p es un invariante del bucle. Supongamos que al comienzo de una ejecución del bucle **while** p es verdadera y se mantiene la condición del bucle; en otras palabras, se supone que $factorial=i!$ y que $i < n$. Los nuevos valores i_{new} y $factorial_{new}$, son $i_{new} = i + 1$ y $factorial_{new} = factorial \cdot (i + 1) = (i + 1)! = i_{new}!$. Como $i < n$, tenemos también que $i_{new} = i + 1 < n$. Por tanto, p es verdadera al

© TODOS LOS DERECHOS RESERVADOS

terminar la ejecución del bucle. Esto muestra que efectivamente *p* es un invariante del bucle.

Ahora consideramos el segmento de programa. Justo antes de entrar en el bucle se cumple que $i = 1 \leq n$ y que *factorial* = 1 = 1! = *i*!, por lo que *p* es verdadera. Como *p* es un invariante del bucle, la regla de inferencia que acabamos de presentar implica que si el bucle **while** termina, termina con *p* verdadera y con $i < n$ falsa. En este caso, al final, *factorial* = *i*! e $i \leq n$ son verdaderas, pero $i < n$ es falsa. En otras palabras, $i = n$ y *factorial* = *i*!=*n*!, como se desea.

Finalmente, necesitamos comprobar que el bucle **while** termina realmente. Al comienzo del programa a *i* se le asigna el valor 1, por lo que tras $n-1$ pasadas del bucle, el nuevo valor de *i* serán y el bucle termina en este punto.

Damos un ejemplo final para demostrar cómo se utilizan varias reglas de inferencia en la verificación de un segmento de programa un poco más grande.

EJEMPLO 4.5.5.

Analicemos cómo se verifica que el programa *S* para calcular el producto de dos enteros es correcto.

procedure *multiplicar* (*m, n*: enteros)

```
S1 { if n < 0 then a := -n
     else a := n

S2 { k := 0
     x := 0

S3 { while k < a
     begin
       x := x + m
       k := k + 1
     end

S4 { if n < 0 then producto := -x
     else producto := x
```

El objetivo consiste en demostrar que una vez que *S* se ejecuta, *producto* tiene el valor *mn*. La demostración de que el programa es correcto se puede hacer partiendo *S* en cuatro segmentos, siendo $S = S_1; S_2; S_3; S_4$, como se muestra en el listado de S. Se puede utilizar la regla de composición para construir esta demostración. Presentamos aquí los argumentos esenciales de la demostración, dejando los detalles para el lector como ejercicio.

© TODOS LOS DERECHOS RESERVADOS

Sea p la afirmación inicial de que m y n son enteros. Entonces, se puede ver que $p\{S_1\}q$ es verdadera cuando q es la proposición $p \wedge (a = |n|)$. Sea r la proposición $q \wedge (k = 0) \wedge (x = 0)$. Se puede verificar fácilmente que $q\{S_2\}r$ es verdadera. Se puede ver también que $x = mk$ y $k \leq a$ es un invariante del bucle en S_3. Además, es fácil ver que el bucle termina tras a iteraciones, con $k = a$, por lo que en este punto $x = ma$. Como r implica que $x = m \cdot 0$ y $0 \leq a$, el invariante del bucle es verdadero antes de entrar en el bucle. Como el bucle termina con $k = a$, se sigue que rS_3s es verdadera, donde s es la proposición $x = ma$ y $a = |n|$. Finalmente, se puede ver que S_4 es verdadera con respecto a la afirmación inicial s y la afirmación final t, donde t es la proposición *producto* = mn.

Reuniendo todos estos argumentos, se ve que $p\{S_1\}q$, $q\{S_2\}r$, rS_3s y $s\{S_4\}t$ son todas verdaderas; se sigue por la regla de composición que $p\{S\}t$ es verdadera. Además, como los cuatro segmentos terminan, S terminará también. Esto verifica que el programa es correcto.

4.5.3. Verificación formal de programas una mirada actual

Tony Hoare en su artículo tilutado "*The Verifying Compiler: a Grand Challenge for Computing Research*" promulgó el "Gran Desafío" para la Investigación en las Ciencias Computacionales: El compilador de Verificación. El Gran Desafío consiste en construir un compilador que utilice razonamiento lógico-matemático automatizado para comprobar, mediante *Verificación formal*, la precisión de los programas que compila. Para lograrlo, el Gran Desafío propone que se incluyan procesos de especificación formal en la Ingeniería de Requisitos.

En la actualidad existen mecanismos para demostrar, matemáticamente, que un sistema o componente software hace lo que tiene que hacer, nada más y nada menos. Estos mecanismos se proporcionan por medio de lenguajes de especificación formal como Z, B, Vienna Development Method VDM, Communicating Sequential Processes CSP, Larch y Formal Development Methodology FDM. Estos lenguajes de especificación se utilizan normalmente para desarrollar modelos, libres de ambigüedades, que describen cómo funcionará el sistema. Esos modelos, de sistemas específicos, se pueden utilizar para aplicarles pruebas matemáticas con el objetivo de asegurar su validez. El resultado de la prueba puede aportar suficiente evidencia de Verificación como para que el desarrollador compruebe si el sistema, cuyo modelo fue especificado formalmente bajo un lenguaje de especificación, es correcto.

A pesar de que los Métodos Formales pueden parecer la solución más eficiente y confiable para la crisis del software, es necesario comprender que no existe tal cosa y que su utilización en la Ingeniería de Software puede incrementar el costo de los productos. Las pruebas de correctitud requieren mucho tiempo para producir una utilidad limitada diferente a la de garantizar exactitud y, por tanto, normalmente se reservan para campos ingenieriles donde los beneficios de tales pruebas permiten que los Métodos Formales agreguen valor a los recursos.

Si los Métodos Formales se reservan para sistemas de seguridad y de misión crítica, entonces se debe identificar claramente cuáles son los sistemas que entran en esas categorías. Fácilmente podrían identificarse sistemas como los aeroespaciales y los militares, donde actualmente se aplican sistemáticamente. Muchos de los sistemas actuales poco a poco y en silencio arrastran procesos críticos de negocio, como los responsables de los procesos electorales, los sistemas

© Todos los derechos reservados

bancarios y muchos otros. Por lo tanto, éstos también deben ingresar en la categoría de sistemas de misión crítica. Una vez que estos sistemas se identifiquen como de misión crítica se puede implementar un análisis de riesgos más apropiado, para comprender la relación entre el costo de verificar formalmente su funcionalidad frente a las consecuencias si se pierde esa funcionalidad debido a una falla del sistema o, en este caso, a un defecto de seguridad.

Metodología correctness-by-construction

C-by-C es una metodología de Ingeniería de Software desarrollada por Praxis High Integrity Systems, que se orienta principalmente hacia la industria aeroespacial y sus sistemas de seguridad crítica. Sin embargo, esta metodología también tiene un impacto significativo en una variedad de industrias, en las que los sistemas de seguridad y de misión crítica son de gran interés. La empresa Praxis HIS ha documentado los logros de aplicación de la metodología de la siguiente manera: Alta productividad: ahorros apercibidos durante las fases de pruebas, Baja cantidad de fallos: quedan muy pocos errores después de la entrega, Software garantizado: emisión de estándares por parte de Praxis para garantizar el software, Bajos costos de soporte: la calidad es fácil de mantener y Clientes satisfechos: se alcanza con éxito los objetivos de los negocios subyacentes del cliente.

La metodología también tiene cinco características distintivas: (1) Utiliza Verificación estática; (2) Utiliza pequeños pasos de diseño verificables; (3) Genera evidencias de certificación y de evaluación como resultado secundario al proceso de desarrollo; (4) Aplica apropiadamente la formalidad y (5) Utiliza notaciones y herramientas adecuadas para el trabajo. La capacidad para eliminar los errores antes de la prueba, o Verificación estática, es lo más complicado de estos principios y características. La capacidad para llevar a cabo esta Verificación, es decir, técnicas de Verificación que no incluyen la ejecución de los programas, depende en gran medida del lenguaje o de la notación en la fase de desarrollo. Praxis recomienda utilizar su herramienta SparkAda, especialmente desarrollada para este propósito y que es un subconjunto del lenguaje de programación Ada, en el que se eliminan ciertas funciones y capacidades para reducir considerablemente el número de posibles formas en que una determinada función se puede llevar a cabo, lo que permite eliminar ciertas ambigüedades. Sin embargo, la capacidad única que proporciona SparkAda, para ayudar al *demostrador de teoremas a verificar formalmente el código*, es las notaciones semánticas estandarizadas. Estas anotaciones semánticas se basan en el lenguaje de especificación formal *Z* y en la teoría axiomática de conjuntos.

Finalmente, los métodos formales, cuando se utiliza adecuadamente, proporcionan una capacidad sin precedentes para crear confianza en la exactitud de un sistema o componente. A través del desarrollo de metodologías como C-by-C y de herramientas como SparkAda, se reducen los costos cada vez más, lo que es ventajoso para la aplicación de estos métodos en el ciclo de vida de la Ingeniería de Software.

La actual dependencia de los sistemas software ha alcanzado un nivel extremadamente alto, al punto que se están re-evaluando los argumentos del pasado acerca de que los métodos formales eran demasiado costosos en tiempo y dinero. El hecho es que cuesta mucho más, en tiempo de inactividad y en mantenimiento, no probar formalmente la corrección de estos sistemas.

© Todos los derechos reservados

Resumen

© Todos los derechos reservados

© Todos los derechos reservados

Algoritmos 5

Objetivos

- Analizar los conceptos y características esenciales de los algoritmos, aplicándolos en el diseño de algoritmos sencillos.
- Valorar la importancia de los algoritmos en la rama de la informática.
- Describir el funcionamiento de una Máquina de Turing, sus características y formas de representación.
- Aplicar el modelo de Máquina de Turing para la solución de problemas sencillos.
- Aplicar los algoritmos recursivos en la solución de problemas.
- Determinar la complejidad de un algoritmo.

© Todos los derechos reservados

5.1. INTRODUCCIÓN

El término algoritmo proviene del nombre del matemático, astrónomo y geógrafo árabe del siglo IX *Al-Khowarizmi* (Rosen, 2012), el cual resaltó la importancia de utilizar procedimientos metódicos para la solución de problemas de álgebra y cálculo numérico. Con el auge de la computación, el concepto de algoritmo fue tomando un significado más amplio que el meramente aplicado a las matemáticas. Los algoritmos son de vital importancia en el área de la informática para la programación, aunque no debe confundirse algoritmo con programa. Un programa es una realización concreta de un algoritmo, por medio de un lenguaje de programación, para resolver un determinado problema; por lo que en la mayoría de los casos resulta más difícil definir correctamente el algoritmo.

Figura 5.1. Proceso de solución computacional de problemas.

En este capítulo se explica cada una de las características presentes en un algoritmo analizando su cumplimiento a través de un ejemplo. Sobre la base del concepto de algoritmo brindado en el capítulo y sus características se muestran varios ejemplos de algoritmos en forma de pseudocódigo. También se describe el funcionamiento y representación de una Máquina de Turing como modelo computacional para el estudio, análisis y comparación de algoritmos. Seguidamente se introduce otra forma de solución de problemas, los algoritmos recursivos, presentándose versiones basadas en la recursividad de algunos de los algoritmos realizados inicialmente. Por último se analiza la complejidad de los algoritmos como método para determinar su eficiencia y realizar comparaciones entre varios algoritmos que resuelven un mismo problema.

5.2. ALGORITMOS

De manera general un algoritmo es un conjunto ordenado de instrucciones bien definidas que permiten realizar una determinada actividad mediante pasos sucesivos. Las recetas de cocina; los manuales de usuario, que muestran indicaciones para usar un equipo o software; las partituras que contienen las instrucciones para tocar una determinada melodía; hasta el método de Gauss para resolver un sistema lineal de ecuaciones constituyen ejemplos de algoritmos utilizados en la solución de un problema determinado.

Un algoritmo debe cumplir con las siguientes características:

- **Entrada:** constituyen los datos necesarios para solucionar el problema planteado y son definidos antes de comenzar el algoritmo.
- **Salida:** es el resultado generado por el algoritmo luego de haber procesado los datos de entrada.
- **Carácter finito:** los algoritmos contienen un número finito de instrucciones que una vez ejecutadas finaliza el algoritmo, dando solución al problema inicial.
- **Precisión:** las instrucciones o pasos deben enunciarse con la mayor claridad posible evitando las ambigüedades.
- **Unicidad:** cada paso intermedio tiene un resultado único, que solo depende de las entradas y los resultados de pasos anteriores.
- **Generalidad:** los algoritmos se diseñan para que sean generales, es decir, que no se apliquen únicamente a un conjunto particular de valores de entradas.

© Todos los derechos reservados

Considere el siguiente al algoritmo para ordenar ascendentemente dos números *a* y *b*:

Paso 1: Si $a > b$, entonces $x = b$, $b = a$, $a = x$.

Paso 2: Mostrar *a*, *b*.

La idea del algoritmo antes presentado es comparar ambos números, en caso de que no estén ordenados se intercambian sus valores. Analizando el cumplimiento de las características necesarias en los algoritmos se puede apreciar que el algoritmo planteado tiene un número finito de pasos pues consta de 2 instrucciones precisas al cabo de las cuales finaliza mostrando el resultado del problema planteado inicialmente. Las entradas se corresponden con los valores de *a* y *b*, mientras que la salida es la lista ordenada de estos dos valores. Es general ya que puede ordenar ascendentemente dos números cualesquiera. Finalmente los resultados de cada paso, correspondientes a los valores de entrada definidos, son únicos independientemente de la computadora o persona que ejecute el algoritmo. Ejemplo:

Para los valores $a = 9$, $b = 4$

En el paso 1, ***a*** > ***b*** por lo que ***x*** toma el valor de ***b*** que es 4 y ***b*** adquiere del valor de ***a*** que es 9, luego se le asigna a la variable a el valor de ***x***. Por tanto al finalizar este paso ***a*** vale 4 y ***b*** vale 9.

En el paso 2 se visualizan los valores de ***a*** y ***b***, ya ordenados.

Los algoritmos pueden formularse en lenguaje natural, pseudocódigo, diagramas Nassi-Shneiderman (NS), diagramas de flujo, o lenguajes de programación. Los algoritmos expresados en lenguaje natural suelen ser ambiguos y extensos, lo que puede evitarse con el uso de pseudocódigos o diagramas de flujo que son más estructurados, facilitando su posterior implementación. Los pseudocódigos, al igual que los diagramas de flujo, son independientes de un lenguaje de programación específico. Los pseudocódigos son una combinación del lenguaje natural con algunos elementos propios de los lenguajes de programación como son las condicionales, los ciclos y las asignaciones, aunque sin regirse por ningún estándar particular. En este capítulo se utilizará principalmente los pseudocódigos en la elaboración de algoritmos, para lo cual se establece la estructura siguiente:

Nombre del algoritmo

Entrada

Salida

Instrucciones

Fin

La primera línea del bloque de instrucciones será un nombre abreviado del algoritmo y entre paréntesis, las variables que identifican las entradas. La última instrucción contendrá la palabra Fin, indicando así la finalización del algoritmo. En las instrucciones se pondrán utilizar los operadores aritméticos "+", "–", "•", "/"; así como los operadores de relación "=", ">", "<", "≠", "≥", "≤" y los operadores lógicos "*and*", "*or*" y "*not*". Se utilizará el signo ":=" para la asignación de valores a las variables. Las condicionales se declaran con se muestra a continuación:

© Todos los derechos reservados

si <condición> ***entonces***

inicio

Instrucción 1

Instrucción 2

...

sino

Instrucción 1

Instrucción 2

...

fin

No es obligatoria la sección ***sino***, en caso de no declararse se asume que si la condición no ocurre no se hace nada y el algoritmo continua con la instrucción posterior al fin de la condicional.

También se pueden utilizar ciclos donde las instrucciones se repiten un determinado número de veces. La cantidad de repeticiones está determinada por los valores enteros <valor inicial> y <valor final>, cada vez que se ejecuta el bloque de instrucciones del ciclo se suma 1 a la variable hasta que su valor sea mayor que <valor final>, en cuyo caso no se ejecutan las instrucciones.

desde *variable* = <valor inicial> ***hasta*** <valor final>

inicio

Instrucción 1

Instrucción 2

...

fin

Existe otro tipo de ciclo en los que las instrucciones se ejecutan mientras se cumpla una determinada condición, a diferencia del ciclo explicado anteriormente, en este caso no se sabe el número de veces que se ejecutará el bloque de instrucción.

mientras <condición>

inicio

Instrucción 1

Instrucción 2

...

© Todos los derechos reservados

fin

También se pueden incorporar comentarios en las líneas de los pseudocódigos con el objetivo de esclarecer el funcionamiento de una instrucción o un bloque de instrucciones. Los comentarios estarán precedidos del símbolo "//", no se ejecutan ni se muestran, sólo constituyen información adicional.

EJERCICIO 5.2.1. ELABORACIÓN DE ALGORITMOS

Escriba en pseudocódigo un algoritmo para:

a) Ordenar ascendentemente dos números *a* y *b*.
b) Identificar el tipo de triángulo a partir de la longitud de sus lados.
c) Obtener la suma de los *n* primeros números naturales.
d) Determinar el mayor de una lista de *n* números.
e) Calcular x^n, teniendo como entradas los números enteros *x* y *n*.
f) Dada una lista de *n* números enteros, insertar un número entero *x* en la posición *p*.

Solución:

a) En este inciso se escribe en pseudocódigo el algoritmo dado en lenguaje natural al principio del epígrafe.

Ordenar ascendentemente dos números.

Entradas: a, b

Salida: a, b ordenados.

ordena2 (a, b)

si a > b *entonces*

inicio

x := b

b := a

a := x

fin

mostrar a, b

Fin

**Nota:* observe que en caso de que a y b estén ordenados inicialmente no se ejecuta la condicional y se pasa directamente a mostrar los valores de las entradas.

b) **Clasificar el triángulo a partir de la longitud de sus lados.**

Entradas: a, b, c (longitud de los lados del triángulo).

Salida: clasificación del triángulo.

© TODOS LOS DERECHOS RESERVADOS

clasifica_triángulo (a, b, c)

si (a = b *and* b = c) *entonces*

mostrar "El triángulo es equilátero"

sino

inicio

si (a = b *or* b = c *or* a = c) *entonces*

mostrar "El triángulo es isósceles".

sino

mostrar "El triángulo es escaleno".

fin

Fin

**Nota:* la idea en este algoritmo es, a partir de las longitudes de los lados entradas, verificar la cantidad de lados iguales y en función de eso mostrar un mensaje con la clasificación correspondiente. Se comenzó confirmando si los tres lados eran iguales (a = b *and* b = c), condición que corresponde a un triángulo equilátero. No es obligatorio empezar verificando si el triángulo es equilátero. En caso de no cumplirse la primera condición se pregunta por cualquiera de las dos condiciones restantes. En este caso se comprueba como segundo paso si el triángulo es isósceles, es decir si tiene dos de sus lados iguales (a = b *or* b = c *or* a = c). En caso de no cumplirse ninguna de las dos condiciones anteriores entonces se puede afirmar que el triángulo es escaleno sin necesidad de verificar de manera explícita la condición (a ≠ b *and* b ≠ c *and* a ≠ c).

c) **Sumar los *n* primeros números naturales.**

Entradas: n

Salida: suma

sumatoria (*n*)

suma := 0

desde i = 1 *hasta* n

suma := suma + i

mostrar suma

Fin

Nota:* este es un algoritmo bastante sencillo, para calcular la suma de los ***n* primeros números naturales se realiza un ciclo donde la variable ***i*** va tomando los valores de 1 hasta el número ***n***, aumentando en 1 después de cada iteración. Se define una variable ***suma*** cuyo valor inicial es 0 y en cada iteración se actualiza sumándole al valor almacenado en ella el valor de ***i*** en esa iteración (suma:= suma + i). Al final del ciclo la variable ***suma*** contendrá el resultado de todas las sumas parciales realizadas en cada iteración, por lo que el último paso es mostrar su valor. La figura 5.2 muestra un ejemplo paso a paso del funcionamiento de este algoritmo.

© Todos los derechos reservados

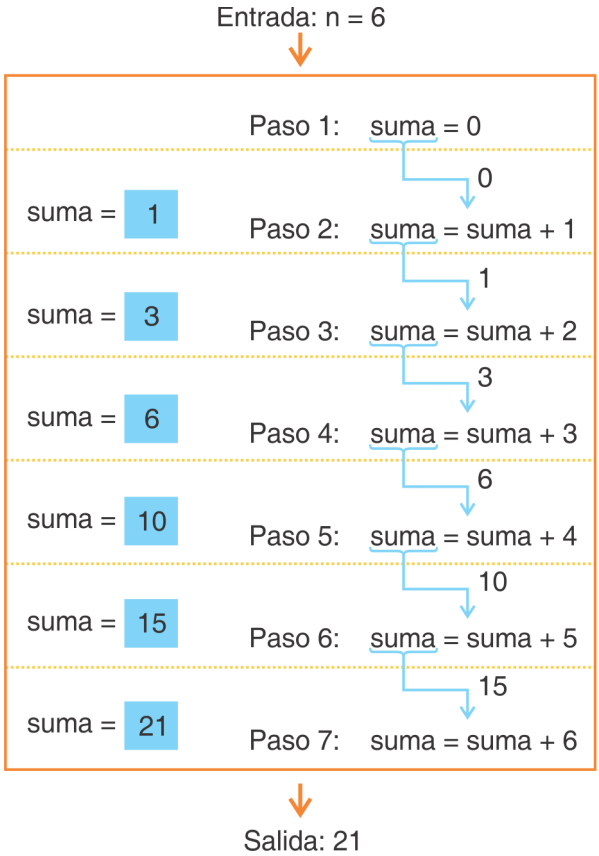


Figura 5.2. Ejemplo del funcionamiento del algoritmo sumatoria.

© Todos los derechos reservados

d) **Determinar el mayor de una lista de *n* números.**

Entradas: lista (*l*) de números con longitud *n*.

Salida: *x*, mayor de los números de la lista.

mayor (*l*)

$x := l_1$

desde i=2 ***hasta*** n

inicio

si $l_i > x$ ***entonces***

$x := l_i$

fin

mostrar x

Fin

En la figura 5.3 se muestra el funcionamiento de este algoritmo tomando como ejemplo la lista: 23, 5, 12, 34, 9, 17.

Figura 5.3. Ejemplo del funcionamiento del algoritmo del *inciso d.*

© Todos los derechos reservados

e) **Calcular potencia x^n.**

Entradas: base *x*, exponente *n.*

Salida: potencia.

potencia (*x*, *n*)

si (x = 0 *and* n = 0) *entonces*

mostrar "indeterminada"

sino

inicio

potencia := 1

si n ≥ 0 *entonces*

desde i = 1 *hasta* n

potencia := potencia • x

sino // en caso de que el exponente (n) sea negativo

desde i = 1 *hasta* (-1 • n) // como *n* es negativo se lleva a positivo multiplicando por (-1)

potencia := potencia • (1/x)

fin

mostrar potencia

Fin

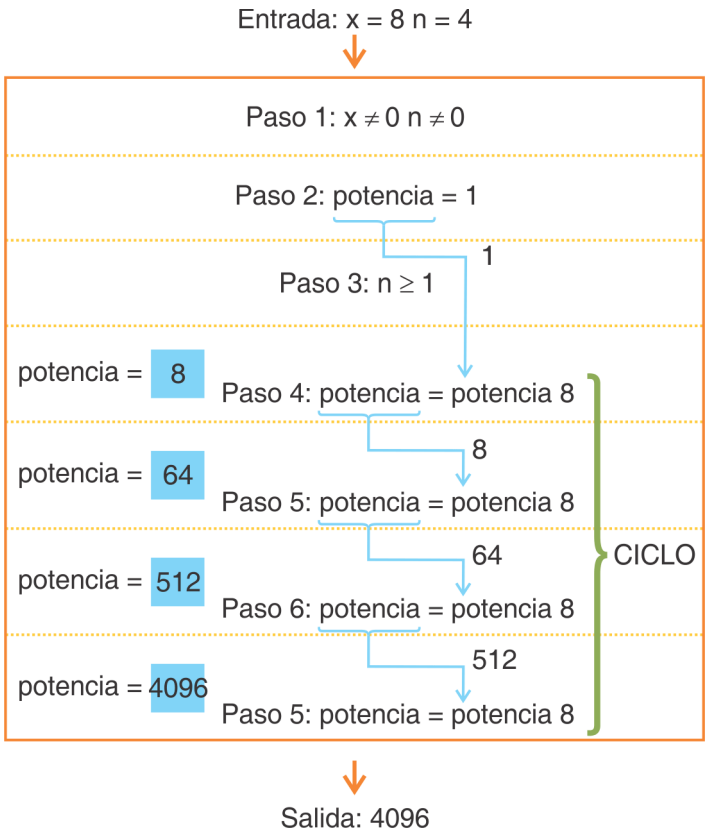


Figura 5.4. Ejemplo del funcionamiento del algoritmo potencia.

f) **Dada una lista de *n* números enteros, insertar un número entero *x* en la posición *p*.**

Entradas: lista de *n* números (*l*), número entero (*x*), posición para la inserción (*p*)

Salida: lista actualizada (*l*)

Insertar_número (*l*, *x*, *p*)

si (p > 0 *and* p ≤ n) *entonces* //asegura que la posición dada se encuentre entre los límites de la lista

inicio

n = n + 1 //incrementa en 1 el tamaño de la lista puesto que se va a insertar un elemento

desde i = n *hasta* p //se recorre la lista de atrás hacia delante hasta la posición *p*

$l_i := l_{i-1}$ //cada elemento posterior a la posición *p* se traslada a la posición inmediata superior

© Todos los derechos reservados

```
lp := x
mostrar l
 fin
sino
mostrar "Posición inválida"
Fin
```

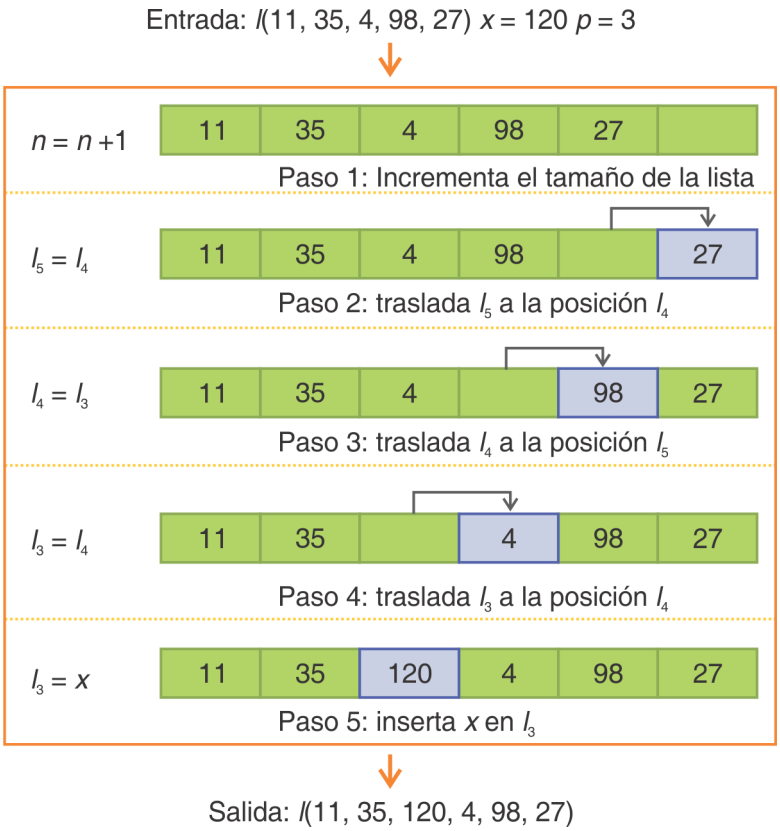


Figura 5.5. Ejemplo del funcionamiento del algoritmo del *inciso f.*

Algoritmo de Euclides

Euclides fue un antiguo matemático griego (325 a. C. – 265 a. C.), al que se conoce como "El Padre de la Geometría". La geometría de Euclides, además de su utilidad matemática y su valor como instrumento de razonamiento deductivo, ha sido de vital importancia en otras áreas del conocimiento; como la Física, la Astronomía, la Química y disímiles ingenierías.

En su obra *Elementos*, Euclides describe un algoritmo muy eficaz para calcular el máximo común divisor (mcd) de dos números, este algoritmo clásico es conocido precisamente con el nombre de

© Todos los derechos reservados

algoritmo de Euclides. Este algoritmo tiene aplicaciones en diversos campos como el álgebra, teoría de números y ciencias de la computación. Dada su gran eficiencia es utilizado, con ligeras modificaciones, en computadoras electrónicas.

El máximo común divisor de dos números enteros a y b es el mayor entero positivo que divide tanto a ***a*** como a ***b*** y se escribe mcd(*a*, *b*). El algoritmo de Euclides se basa en el hecho de que el máximo común divisor de dos números ***a*** y ***b*** lo es también de ***b*** y ***r***, donde ***r*** es el resto de dividir ***a*** entre ***b***; en lenguaje matemático: mcd(*a*, *b*) = mcd(*b*, *r*). Es importante tener en cuenta que:

- ***a*** y ***b*** no pueden ser ambos cero.
- El máximo común divisor de cualquier número entero ***n*** y 0 es el propio número, es decir, mcd(*n*, 0) = *n*.

El algoritmo consiste en reducir el problema inicial de calcular *mcd*(a, b) al cálculo de *mcd*(b, r) que son números más pequeños. Luego se divide ***b*** y ***r*** para obtener un nuevo resto r_1 con el que se reduce *mcd*(b, r) a *mcd*(r, r_1). Este procedimiento se repite sucesivamente hasta obtener resto cero (r_n = 0) en cuyo caso *mcd*(r_{n-1}, 0) = r_{n-1}, luego como *mcd*(a, b) = *mcd*(b, r) = *mcd*(r, r1) = ...= *mcd*(r_{n-1}, 0) = r_{n-1}, entonces *mcd*(a, b) = r_{n-1}. Este método requiere que ***a*** sea mayor que ***b*** (a > b).

En la figura 5.6 se muestra el funcionamiento paso a paso del algoritmo tomando como ejemplo los números 504 y 396.

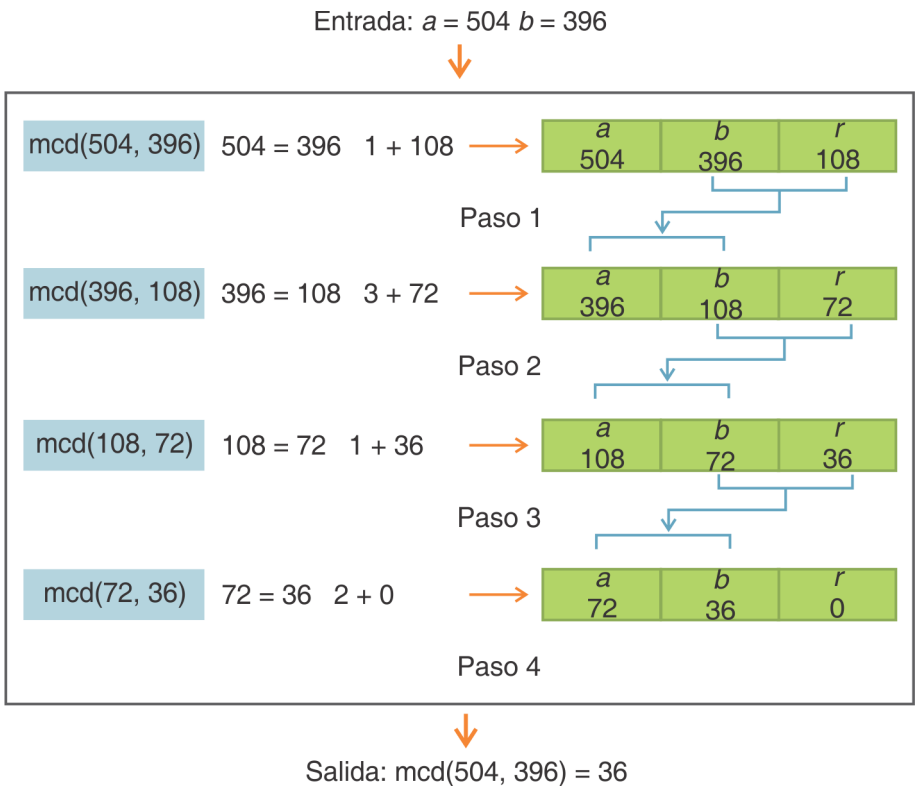


Figura 5.6. Pasos del algoritmo de Euclides para los números 504 y 396.

© Todos los derechos reservados

A continuación se muestra el pseudocódigo del algoritmo de Euclides, en el se utilizará la expresión "*x mod y*" para indicar el resto de la división de *x* entre *y*:

```
Algoritmo de Euclides
Entradas: a, b
Salida: mcd
mcd (a, b)
    si a < b entonces //asegura que a sea mayor que b, en caso contrario se
    intercambian los valores.
    inicio
        x := b
        b := a
        a := x
    fin
mientras b ≠ 0
    inicio
        r := a mod b
        a := b
        b := r
    fin
mostrar a
Fin
```

5.3. MÁQUINA DE TURING

Para el estudio de los algoritmos es necesario establecer un modelo computacional, que no es más que un marco en el cual estudiar, probar, analizar y comparar algoritmos. El más básico de los modelos computacionales se denomina *máquina de Turing* propuesto en 1936 por el matemático, criptógrafo e informático teórico inglés Alan Turing[1] . A pesar de su sencillez la máquina de Turing tiene la misma capacidad de cómputo que cualquier computador digital por moderno que este sea.

1. Alan Mathison Turing es considerado uno de los padres de la Ciencia de la computación y precursor de la informática moderna. Formalizó los conceptos de algoritmo y computación, también contribuyó al desarrollo de la Inteligencia Artificial. En su honor, la *Association for Computing Machinery (ACM)* otorga anualmente el "Premio Turing" a personas destacadas por sus aportes en el campo de las ciencias computacionales, que es considerado el equivalente al Premio Nobel en el mundo de la computación.

© Todos los derechos reservados

La máquina de Turing consta de un cabezal lector/escritor y una cinta infinita en ambas direcciones, dividida en espacios denominados celdas. El cabezal lector/escritor apunta a una posición de la cinta y sobre él se pueden realizar las operaciones siguientes:

- Mover el cabezal lector/escritor una posición hacia la derecha (D).
- Mover el cabezal lector/escritor una posición hacia la izquierda (I).
- Mantener el cabezal lector/escritor en la posición actual (M)

En cada paso el cabezal lee un símbolo de la cinta y escribe un nuevo valor o el mismo. En este proceso también se tiene en cuenta el estado *q* en que está la máquina en un momento dado. En la figura 5.7 se muestra un esquema ideal de la máquina de Turing.

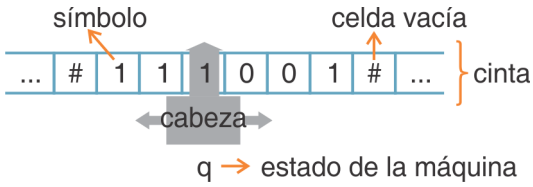


Figura 5.7. Esquema de una máquina de Turing.

Se considera que la cinta siempre comienza con un símbolo en blanco (marca de inicio), seguido por la cadena que se quiere procesar, y el resto de la cinta está ocupado por símbolos en blanco. Se supone además que el cabezal se encuentra inicialmente ubicado en el primer símbolo de la cadena (sin contar el símbolo en blanco que marca el inicio).

Una máquina de Turing se puede representar de dos formas:

- Representación Formal.
- Diagrama de estados.

Representación formal

Una Máquina de Turing se puede representar por una tupla de la forma MT = {Q, Σ Γ, q_0, F, θ} donde:

Q: Conjunto finito de estados.

Σ: conjunto finito no vacío de símbolos denominado alfabeto de entrada. Ejemplo: Σ= {0,1}, Σ_1= {a, b, c}

Γ: conjunto finito no vacío de símbolos denominado alfabeto de la cinta. ($\Sigma \subset \Gamma$, # $\in \Gamma$)
**Nota:* el símbolo # se denomina blanco y representa el espacio vacío.

q_0: estado inicial. ($q_0 \in$ Q)

F: Conjunto de estados finales. (F $\subseteq$ Q)

© Todos los derechos reservados

θ: Función de transición. θ (Q, Γ) = (Q, Γ, { I, D, M })
*Nota: la función de transición funciona como reglas que utilizan el estado actual, y el símbolo leído de la cinta, para decidir qué acción realizar. Cada regla (q_0, S_0) = (q_1, S_1, [I | D | M]) es interpretada de la siguiente forma: "si el estado actual es q_0 y el símbolo leído por el cabezal es S_0, entonces escribir el símbolo S_1 en la cinta, mover el cabezal hacia [izquierda | derecha | mantenerse], y pasar al estado q_1".

Diagrama de estados

El diagrama de estados brinda una representación gráfica de la máquina de Turing utilizando un grafo dirigido donde los estados constituyen nodos encerrados en círculos con su identificador en el interior del círculo.

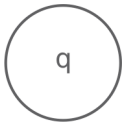


Figura 5.8. Representación de un estado.

Se especifica cuál es el estado inicial con una flecha o con palabras, y se resaltan los estados finales encerrándolos en dos círculos.

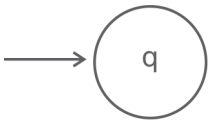


Figura 5.9. Representación del estado inicial.

Figura 5.10. Representación de un estado final.

Se dibujan aristas dirigidas (flechas) de un nodo a otro nodo, no necesariamente distintos, y se etiquetan estas aristas para representar la función de transición. En las figuras 5.11 y 5.12, S_1 representa el símbolo leído y S_2 el símbolo escrito.

© Todos los derechos reservados

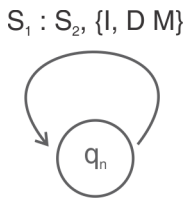


Figura 5.11. Representación de una transición sin cambio de estado.

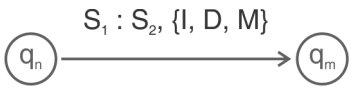


Figura 5.12. Representación de una transición de un estado a otro.

© Todos los derechos reservados

Ejemplo 5.3.1. Máquina de Turing

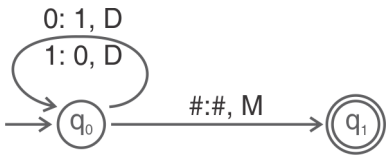
Construya una máquina de Turing que tenga las siguientes características:

a) Dada una cadena de longitud mayor que cero de caracteres binarios, transforme los 1s en 0s y viceversa (operador negación).

b) Dada una cadena sobre el alfabeto {a, b}, encuentre las subcadenas "aa" y las cambie por "bb" y las subcadenas "bb" las cambie por "aa". La cadena finaliza cuando se lee un espacio en blanco (#). El cabezal se encuentra sobre el primer carácter de la cadena.

Solución:

a)

Representación formal	Diagrama de estados
MT = {Q, Σ, Γ, q_0, F, θ} Q = {q_0, q_1} Σ = {0, 1} Γ = {0, 1, #} F = {q_1} θ: (q_0, 0) = (q_0, 1, D) (q_0, 1) = (q_0, 0, D) (q_0, #) = (q_1, #, M)	

La figura 5.13 ilustra paso a paso el funcionamiento de esta máquina de Turing utilizando como entrada la cadena de caracteres 01101.

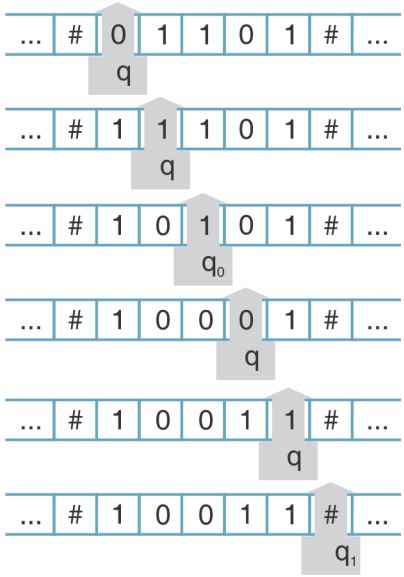
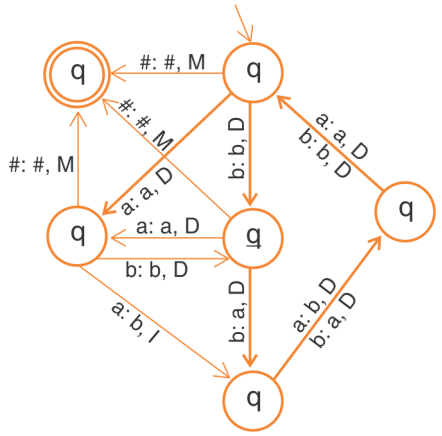


Figura 5.13. Ejemplo del funcionamiento de la máquina de Turing del inciso *a* del ejercicio 5.3.1

b)

Representación formal	Diagrama de estados
MT = {Q, Σ, Γ, q_0, F, θ} Q = {q_0, q_1, q_2, q_3, q_4, q_5} Σ = {a, b} Γ= {a, b, #} F = {q_5} θ: (q_0, a) = (q_1, a, D) (q_0, b) = (q_2, b, D) (q_0, #) = (q_5, #, M) (q_1, a) = (q_3, b, I) (q_1, b) = (q_2, b, D) (q_1, #) = (q_5, #, M) (q_2, a) = (q_1, a, D) (q_2, b) = (q_3, a, I) (q_2, #) = (q_5, #, M) (q_3, a) = (q_4, b, D) (q_3, b) = (q_4, a, D) (q_4, a) = (q_0, a, D) (q_4, b) = (q_0, b, D)	

- q_0 es el estado inicial que indica el comienzo de la cadena.
- El estado q_1 indica que se encontró la primera “a” de un posible par “aa”.
- El estado q_2 indica que se encontró la primera “b” de un posible par “bb”.

© Todos los derechos reservados

- El estado q_3 indica que se encontró el segundo símbolo del par ya sea una “a” para el par “aa” o una “b” para el par “bb”. Luego de leer el símbolo se cambia el valor del símbolo y se vira hacia atrás para cambiar el primer símbolo del par.
- El estado q_4 indica la terminación de la transformación de un par.
- q_5 es el estado final que indica que se terminó la cadena.

La figura 5.14 ilustra paso a paso el funcionamiento de esta máquina de Turing utilizando como entrada la cadena de caracteres “abaaa”.

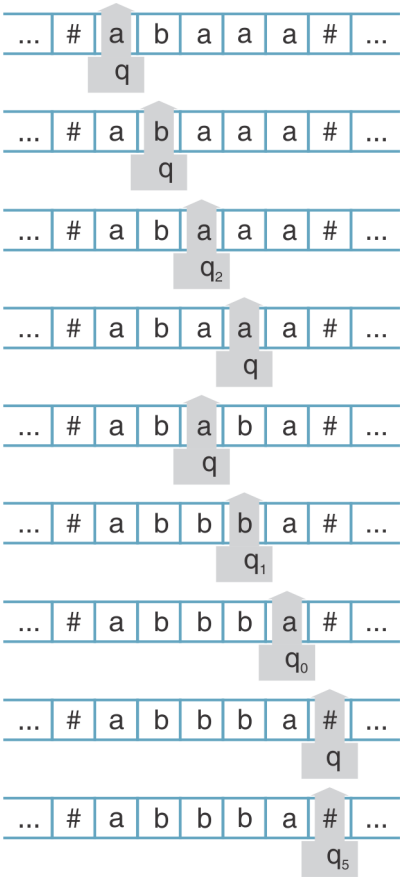


Figura 5.14. Ejemplo del funcionamiento de la máquina de Turing del inciso *b* del ejercicio 5.3.1.

© Todos los derechos reservados

5.4. ALGORITMOS RECURSIVOS

En la cultura popular se utiliza la frase "*Divide y vencerás*" para hacer referencia a la habilidad de resolver un problema complejo dividiéndolo en partes más simples. Este refrán da nombre a una técnica de diseño de algoritmos de gran importancia en la programación, una forma de implementarla se basa en el uso de la recursividad.

Un ***algoritmo recursivo*** es aquel en el cual se soluciona un problema descomponiéndolo en uno o varios pequeños subproblemas similares al original. Cada subproblema, a su vez, se descompone nuevamente hasta que el proceso produzca subproblemas lo suficientemente sencillos como para que sean resueltos directamente. Por último, las soluciones a cada uno de los subproblemas se combinan para obtener una solución del problema original. De esta forma se pueden identificar dos partes fundamentales dentro de los algoritmos recursivos, el caso base y el caso recurrente:

- ***Caso base:*** para todos los algoritmos recursivos tiene que existir al menos un caso donde el problema no sea resuelto de manera recursiva, sino que se obtiene la solución directamente.
- ***Caso recurrente:*** es la parte del algoritmo que resuelve el problema haciendo una o más llamadas al mismo método pero con datos más pequeños. Es importante que todas las llamadas recursivas progresen hacia el caso base.

La recursividad permite resolver cualquier tipo de problema de una forma más elegante y natural, aunque diseñar algoritmos de este tipo requiere práctica e ingenio. A continuación se muestra un ejemplo donde se presentan dos algoritmo para el cálculo del factorial de un número, uno se realiza por vía iterativa y el segundo aplica la recursividad.

```
Factorial de n
Entradas: n
Salida: factorial
factorial (n)
    factorial := 1
    si n < 0 entonces
        mostrar  "El factorial se define solo para los números positivos"
    sino si (n
    > 1) entonces
        inicio
        desde i = 2 hasta n
        factorial := factorial • i
         fin
```

© Todos los derechos reservados

mostrar factorial

Fin

Se debe tener en cuenta que el factorial se calcula solo para los números positivos y que el factorial de 0 y de 1 es igual a 1 (*0!=1; 1!=1*). El algoritmo planteado es bastante sencillo, se define una variable ***factorial*** cuyo valor inicial es 1, para números mayores que 1 se realiza un ciclo donde la variable ***i*** va tomando los valores desde 2 hasta el número ***n***, aumentando en 1 después de cada iteración. En cada iteración se actualiza la variable factorial multiplicándole, al valor almacenado en ella, el valor de ***i*** en esa iteración (factorial := factorial • i). Al final del ciclo la variable ***factorial*** contiene el resultado de todas las multiplicaciones parciales realizadas en cada iteración, por lo que el último paso es mostrar su valor. En caso de que el número de entrada ***n*** fuese 0 o 1 no se ejecuta el ciclo, por tanto al mostrar la variable factorial esta contendrá el valor con que se inicializó (factorial := 1). La figura 5.15 muestra un ejemplo paso a paso del funcionamiento de este algoritmo.

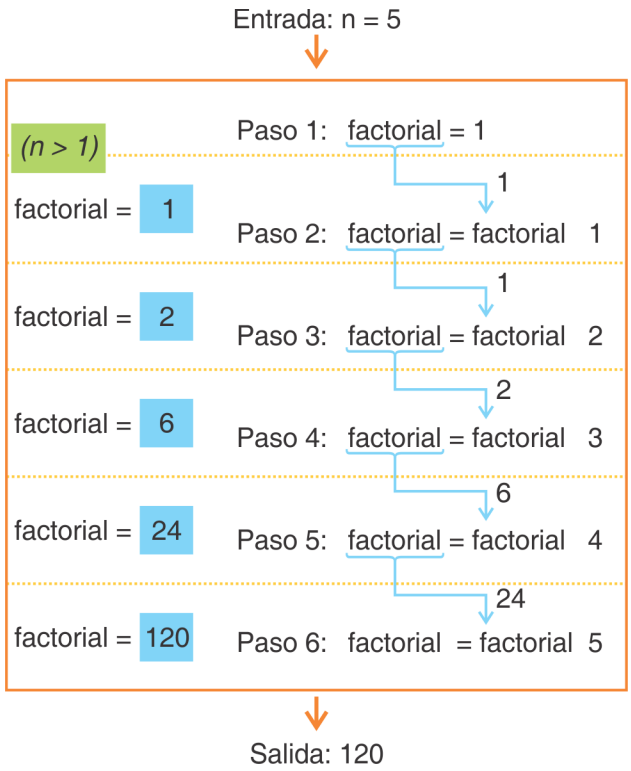


Figura 5.15. Ejemplo del funcionamiento del algoritmo iterativo *factorial*.

En la versión recursiva del algoritmo factorial, el procedimiento se llama a sí mismo con un argumento cada vez menor hasta llegar al caso base 0!=1. Es decir:

$n! = n \bullet (n - 1)!$
$(n - 1)! = n \bullet (n - 2)!$
$(n - 2)! = n \bullet (n - 3)!$

© Todos los derechos reservados

.
.
.
0! = 1

Luego, como se explicó anteriormente, se combinan las soluciones de los subproblemas ((n – 1)!, (n – 2)! ... 0!) para obtener la solución del problema original (n!). A continuación se muestra el pseudocódigo de este algoritmo recursivo y una imagen explicativa de su funcionamiento:

Factorial de *n*

Entradas: n

Salida: factorial

factorial (*n*)

si n < 0 ***entonces***

mostrar "El factorial se define solo para los números positivos"

sino

si (n = 0 ***or*** n = 1) ***entonces***

mostrar 1

sino

mostrar n • factorial(n-1) // esta instrucción se ejecuta solo cuando n > 1

Fin

© TODOS LOS DERECHOS RESERVADOS

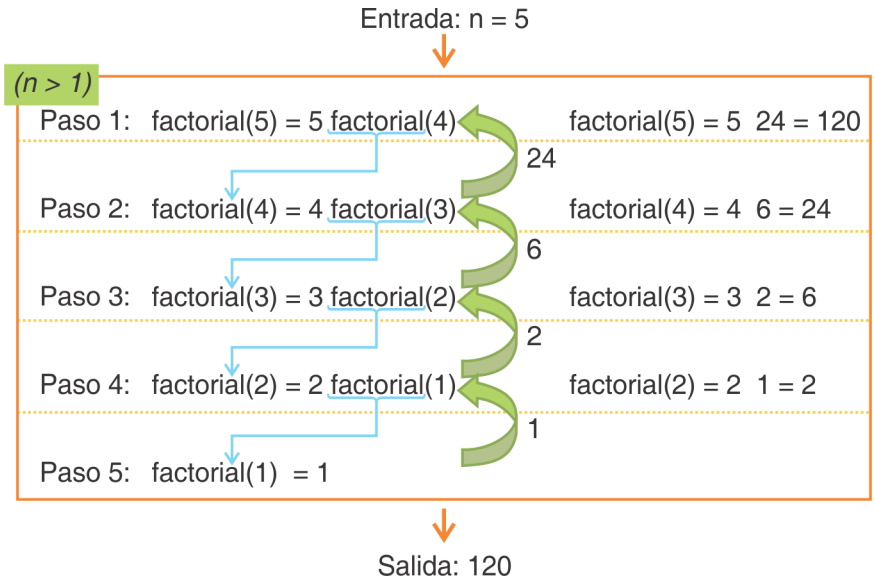


Figura 5.16. Ejemplo del funcionamiento del algoritmo recursivo *factorial*.

En el procedimiento iterativo se obtiene un resultado en cada iteración el cual se va guardando en la variable factorial, mientras que en la versión recursiva no se obtienen resultados hasta tanto no se llegue al caso base luego en recorrido inverso se va solucionando cada cálculo pendiente con los valores obtenidos en cada paso anterior partiendo del caso base.

Luego de haber estudiado los algoritmos recursivos se puede apreciar que el algoritmo de Euclides presentado en la sección 5.2 está explicado de manera recursiva, sin embargo el pseudocódigo presentado no utiliza este concepto. A continuación se muestra una variante más conveniente, simple y elegante de calcular el máximo común divisor de dos números, utilizando la recursividad.

Algoritmo de Euclides recursivo.

Entradas: *a, b*

Salida: *mcd*

mcd (*a, b*)

si a < b *entonces* //asegura que a sea mayor que b, en caso contrario se intercambian los valores.

inicio

x := b

b := a

a := x

fin

si b = 0

mostrar *a*

sino

mostrar mcd (b, a *mod* b)

Fin

5.5. COMPLEJIDAD DE UN ALGORITMO

La eficiencia de un algoritmo se suele expresa en términos de su ***complejidad computacional***, la cual estudia los recursos que consume el algoritmo en su ejecución. Los recursos comúnmente analizados son el ***tiempo*** y el ***espacio en memoria*** empleados para resolver el problema. El tiempo se determina mediante una aproximación a la cantidad y tipo de operaciones (pasos) que la computadora debe realizar en la ejecución del algoritmo, mientras que el espacio es una aproximación a la cantidad de memoria utilizada.

© Todos los derechos reservados

Complejidad temporal

Determinar con precisión el tiempo que emplea un programa en resolver un problema determinado depende de varios factores: computadora utilizada, forma de representación de los datos, forma en que el programa se traduce en instrucciones de máquina, entre otros. Sin embargo se puede obtener una aproximación de costo de tiempo de ejecución de un programa analizando el tiempo del algoritmo implementado.

Habitualmente no es tan importante el tiempo exacto que demora un algoritmo en ejecutarse, sino que resulta más relevante conocer su tasa de crecimiento en función del tamaño o la cantidad de datos de entrada. Con este fin se define el ***orden*** del algoritmo utilizando la ***notación O*** (*Big-O notation*, en inglés), lo que permite no solo abstraerse de factores de hardware y software, sino que además asume que todas las operaciones realizadas en el algoritmo consumen el mismo tiempo, lo que simplifica el análisis considerablemente. Determinar el orden de un algoritmo es un elemento importante en la comparación de distintos algoritmos que resuelven un mismo problema y decidir cuál de ellos es más eficiente en función del crecimiento del tamaño de los datos de entrada.

Una función $f(x)$ pertenece a $O(g(x))$ cuando existe una constante positiva C tal que a partir de un valor x_0, $f(x)$ no sobrepasa a $Cg(x)$. Es decir que salvo por constantes y un número finito de excepciones, f está acotada superiormente por g; la figura 5.17 ilustra esta observación. De esta forma la notación $O(g(x))$ es útil para acotar el tiempo en el peor de los casos posibles para tamaños muy grandes de la entrada n.

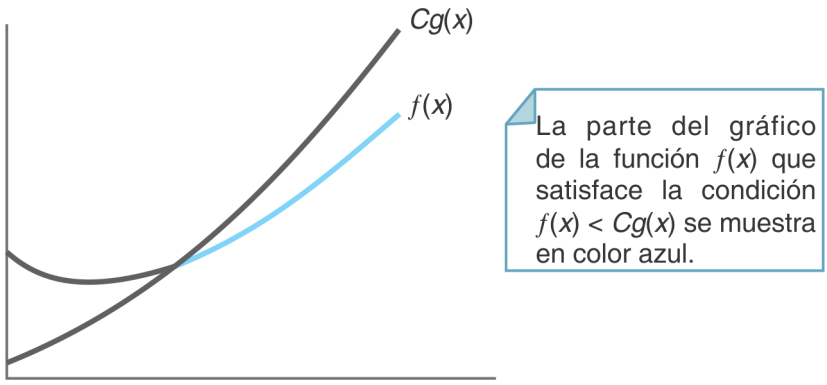


Figura 5.17. Representación gráfica de una función $f(x)$ de orden $O(g(x))$.
Fuente: modificado a partir de Rosen, 2012.

Combinación de funciones

- Suma:

  Si $f_1(x)$ es de orden $g_1(x)$ y $f_2(x)$ es de orden $g_2(x)$ entonces el orden de $f_1(x) + f_2(x)$ es el máximo de los órdenes (max (g_1, g_2)). En caso de que $f_1(x)$ y $f_2(x)$ sean ambas de orden $g(x)$ entonces $f_1(x)$ y $f_2(x)$ es de orden $g(x)$, pues max $(g, g) = g$.

© Todos los derechos reservados

- Multiplicación:

  Si $f_1(x)$ es de orden $g_1(x)$ y $f_2(x)$ es de orden $g_2(x)$ entonces el orden de $f_1(x) \bullet f_2(x)$ es la multiplicación de los órdenes $(g_1 \bullet g_2)$.

  Otra propiedad muy útil de la *notación O* define que si existe el $\lim_{x\to\infty} \frac{f(n)}{g(n)} = k$, dependiendo de los valores que tome k se tiene que:

  a) Si $k \neq 0$ y $k < \infty$ entonces $O(f) = O(g)$.

  b) Si $k = 0$ entonces $f \in O(g)$ y $g \notin O(f)$.

Existen otras notaciones útiles en el análisis de algoritmos:

- La ***notación omega***, Ω: la expresión $f(x)= \Omega\ (g(x))$ significa que, salvo por constantes y un número finito de excepciones, la función f está acotada inferiormente por g.
- La ***notación theta***, Θ: se aplica cuando la función $f(x)$, que determinar el tiempo de ejecución de un algoritmo, crece con la misma razón que una función $g(x)$. De manera que, $f(x)= \Theta\ (g(x))$ expresa que, salvo por constantes y un número finito de excepciones, la función $f(x)$ está acotada superior e inferiormente por $g(x)$, es decir, que $f(x) = O(g(x))$ y $f(x) = \Omega\ (g(x))$.

Es necesario aclarar que el uso del signo "=" para expresar el orden de un algoritmo, aunque es muy común, no es exactamente correcto ya que O, Ω y Θ se definen como conjuntos por lo que en lugar de una igualdad se debería usar el signo "$\in$".

Para determinar el orden de una función se debe tener en cuenta:

- Seleccionar el menor orden posible de la función. Ejemplo: "2*n* es de orden *n*" en vez de "2*n* es de orden n^2".
- Reducir al máximo la expresión utilizada para definir el orden de la función, eliminando las constantes. Ejemplo: "5*n* + 8 es de orden *n*" en vez de "5*n* + 8 es de orden 5*n*".
- Si una determinada función $f(x)$ es de orden $\log_a n$ también es de orden $\log_b n$ para todo **a** y **b**, por lo que no es necesario especificar la base siendo entonces $f(x)$ de orden *log n*.

Se pueden obtener tres funciones para medir el tiempo de ejecución de un algoritmo dado, correspondientes a su caso mejor, medio y peor. Para cada una de ellas se pueden definir tres cotas asintóticas de crecimiento (O, Ω, Θ). Para simplificar, dado un algoritmo se determinará su orden de complejidad para el peor caso utilizando únicamente la ***notación O***. Es decir, el orden de complejidad para el peor caso es $O(g(x))$, que indican que el tiempo de ejecución es de orden a lo más $g(x)$. De forma análoga su orden de complejidad para el mejor caso es $\Omega\ (g(x))$ y su tiempo de ejecución para el mejor caso es de orden por lo menos $g(x)$. Mientras que $\Theta\ (g(x))$ se utilizará para indicar el orden exacto de complejidad del algoritmo. A continuación se explica qué se entiende por caso mejor y caso peor.

Caso mejor: se corresponde con el menor de los tiempos requeridos por el algoritmo para ejecutarse con cada una de las instancias del problema.

© Todos los derechos reservados

Caso peor: se corresponde con el mayor de los tiempos requeridos por el algoritmo para ejecutarse con cada una de las instancias del problema.

Para ilustrar estos conceptos se utilizará el algoritmo presentado en el inciso *d* del ejercicio 5.2.1 teniendo en cuenta la cantidad de veces que se ejecuta la instrucción "$l_i := l_{i-1}$" dentro del ciclo:

Dada una lista de *n* números enteros, insertar un número entero *x* en la posición *p*.

Entradas: lista de *n* números *(l)*, número entero *(x)*, posición para la inserción *(p).*

Salida: lista actualizada *(l).*

Insertar_número (*l, x, p*)

si (p > 0 *and* p ≤ n) *entonces*

inicio

n = n + 1

desde i = n *hasta* p

$l_i := l_{i-1}$

$l_p := x$

mostrar l

fin

sino

mostrar "Posición inválida"

Fin

El mejor caso ocurre cuando la posición de inserción coincide con el final de la lista (p = n) en cuyo caso se realiza una sola vez.

El peor caso se observa cuando la posición de inserción coincide con la primera posición de la lista, en ese caso se realizan n asignaciones (desplazamientos hacia la posición siguiente).

Algunas reglas prácticas generales:

- Las instrucciones elementales (IE) se consideran de orden constante, es decir, de orden 1. Ejemplo: operaciones aritméticas con datos de tamaño constante, asignaciones, lectura o escritura de datos, comparaciones, llamadas a funciones o procedimientos y retorno desde ellos.
- El tiempo de ejecución de una secuencia consecutiva de instrucciones se calcula sumando los tiempos de ejecución de cada una de las instrucciones, por lo que aplicando la regla de la suma, sería el máximo de los órdenes de cada instrucción.
- El tiempo necesario para ejecutar la condicional "*si* <condición> *entonces* <A> *sino* <B>" es el tiempo requerido para evaluar la condición más el tiempo máximo de ejecución de los

© Todos los derechos reservados

bloques de instrucciones A y B. El razonamiento subyacente es que la condición siempre se ejecuta por lo que se debe considerar su tiempo de ejecución, luego se ejecuta el bloque de instrucciones A o el bloque B, nunca los dos; en el peor de los casos se ejecutará el de mayor tiempo, por eso se debe tomar, entre los dos bloques, el máximo tiempo de ejecución.

- Para el ciclo mientras el tiempo de ejecución es la suma del tiempo necesario para evaluar la condición más el número de iteraciones multiplicado por la suma del tiempo de la condición y el de las instrucciones:

  T= T(Condición) + iteraciones * (T(Condición) + T(Instrucciones))

  En otras palabras, la evaluación de la condición se ejecuta al menos una vez, luego si esta se cumple se procede a ejecutar una iteración; en cada iteración se ejecutan las instrucciones del ciclo y otra vez la condición para determinar si continúa con otra iteración o se detiene el ciclo. Tanto el tiempo de la condición como el de la ejecución de las instrucciones pueden variar en cada iteración, lo que debe tenerse en cuenta para el cálculo.

  **Nota:* para calcular el tiempo de ejecución de otro tipo de sentencia repetitiva basta expresarla como un ciclo mientras. Ejemplo:

Tabla 5.1: Transformación de un ciclo "desde" en un ciclo "mientras".

Ciclo desde	Ciclo mientras
desde i := 1 **hasta** n <Instrucciones> ***fin***	i := 1 **mientras** i < n <Instrucciones> i := i + 1 ***fin***

Transformación

- El orden de una llamada a un método, procedimiento o función, Método (P1, P2,..., Pn), es el de ejecutar la llamada (orden 1), más el orden de cada uno de los parámetros P1, P2,..., Pn, sumado al orden del bloque de instrucciones que se ejecutan dentro del Método.

  T = 1 + T(P1) + T(P2) + ... + T(Pn) + T(Método).

  ** Las asignaciones con varias llamadas a funciones deben sumar los órdenes de cada llamada.*

EJEMPLO 5.5.1. COMPLEJIDAD DE ALGORITMOS

De los algoritmos desarrollados en el ejercicio 5.2.1, determine la complejidad de los siguientes:

© TODOS LOS DERECHOS RESERVADOS

a) Obtener la suma de los *n* primeros números naturales.

b) Determinar el mayor de una lista de *n* números.

Solución:

a) ***sumatoria*** (*n*)

```
        suma := 0
        desde i = 1 hasta n
        suma := suma + i
        mostrar suma
    Fin
```

b)

Determinar el mayor de una lista de n números (peor caso)		
1.	***mayor*** (*l*)	
2.	x := l_1	2 IE
3.	i := 2	1 IE
4.	***mientras*** i < n	T(condición) = 1 IE
5.	***inicio***	-----
6.	***si*** l_i > x ***entonces***	T(condición) = 2 IE
7.	***inicio***	-----
8.	x := l_i	2 IE
9.	***fin***	T(si) = 2 + max(2, 0) = 4 IE
10.	i := i + 1	2 IE
11.	***fin***	T(mientras) = 1 + (n – 1) • (6 + 1) = 7n – 6 IE
12.	***mostrar*** x	1 IE
13.	Fin	T(mayor) = 3 + 7n – 6 + 1 = 7n – 2 IE

Explicación en base a las reglas presentadas anteriormente:

- En la línea 2 se realizan 2 IE: el acceso al primer elemento de la lista y la asignación a la variable x.
- En la línea 3 se realiza 1 IE: la asignación.
- En la condición del ***mientras*** (la línea 4) se realiza 1 IE: la comparación.
- En la condición del ***si*** (la línea 6) se realizan 2 IE: el acceso a una posición de la lista y la comparación. Luego T(***si***) = T(Condición)+ max(T(***entonces***), T(***sino***)), como no hay un bloque de instrucciones para el ***sino***, se obtiene que T(***si***) = 2 + max(2, 0) = 4 IE.
- En la línea 10 se realizan 2 IE: la suma y la asignación.
- Entonces el bloque de instrucciones contenidas en el ciclo ***mientras*** (desde la línea 4 hasta la 11) realiza 6 IE: las 4 del ***si*** más las 2 de la línea 10.
- Analizando el ciclo se puede observar que la variable de control *i* se inicializa en 2 y en cada iteración incrementa su valor en una unidad hasta que se iguala a la longitud de la lista (n), por lo que el ciclo ***mientras*** realiza (n – 1) iteraciones.

© TODOS LOS DERECHOS RESERVADOS

- Luego T(***mientras***) = T(Condición) + Iteraciones • (T(Instrucciones) + T(Condición)) = 1 + (n – 1) • (6 + 1) = 7n – 6 IE.
- En la línea 12 se realiza 1 IE: mostrar la salida del algoritmo.
- Entonces el algoritmo realiza 2 IE de la línea 2, más 1 IE de la línea 3, más 7n – 6 IE del ***mientras***, más 1 IE de la línea 12, para un total de 7n – 2 IE.

La cantidad de operaciones se calculó para el peor de los casos f(n)= 7n – 2, la cota superior para esta función se obtiene eliminando los factores constantes, obteniéndose que f(x) es O(n).

Note que en el mejor de los casos, en que nunca se ejecute la instrucción x := l_i dentro de la condicional, aun se debe recorrer la lista completa realizando las (n – 1) comparaciones para determinar si se cumple la condición l_i > x. Por lo que se puede demostrar que f(x) es Ω (n):

Determinar el mayor de una lista de n números (mejor caso)		
1.	***mayor*** (*l*)	
2.	x := l_1	2 IE
3.	i := 2	1 IE
4.	***mientras*** i < n	T(condición) = 1 IE
5.	***inicio***	-----
6.	***si*** l_i > x ***entonces***	T(condición) = 2 IE
7.	***inicio***	-----
8.	x := l_i	no se ejecuta: 0 IE
9.	***fin***	T(si) = 2 + max(0, 0) = 2 IE
10.	i := i + 1	2 IE
11.	***fin***	T(mientras) = 1 + (n – 1) • (4 + 1) = 5n – 4 IE
12.	***mostrar*** x	1 IE
13.	Fin	T(mayor) = 3 + 5n – 4 + 1 = 5n IE

Luego como que f(x)=O(g(x)) y f(x)= Ω(g(x)) entonces se puede afirmar que f(x)=Θ (n).

De manera general los polinomios en n de grado k, donde cada a_i representa una constante, cumplen la regla siguiente para la determinación de las cotas asintóticas:

$f(x)= a_k n^k + a_{k-1} n^{k-1} + ... + a_1 n + a_0$

$f(x)= O(n^k)$

$f(x)= \Omega(n^k)$

$f(x)= \Theta(n^k)$

Es decir, las funciones polinomiales tienen como orden el término de mayor grado (potencia). De esta forma se prescinde de los términos de orden menor y de los factores constantes, ya que la suma devuelve el máximo de los órdenes de los sumandos, considerando cada término como una función independiente.

Por tanto otra forma de determinar las cotas del ejercicio es siguiendo el criterio anterior, dado que $f(x) = 7n - 2$ es una función polinomial, por lo que:

$f(x)= O(n)$

$f(x)= \Omega(n)$

$f(x)= \Theta(n)$

© Todos los derechos reservados

La tabla 5.2 muestra algunos de los órdenes utilizados en el análisis de algoritmos, organizados de forma creciente, donde *a* representa una constante y *n* el tamaño de la entrada.

Tabla 5.2: Listado de órdenes.

Notación	Terminología
Θ (1)	orden constante.
Θ (log *log n*)	orden sublogarítmico.
Θ (log n)	orden logarítmico.
Θ (n)	orden lineal.
Θ (n log n)	orden lineal logarítmico.
Θ (n^2)	orden cuadrático.
Θ (n^3)	orden cúbico.
Θ (n^a)	orden polinomial.
Θ (a^n) n>1	orden exponencial.
Θ (n!)	orden factorial.

© Todos los derechos reservados

Resumen

© Todos los derechos reservados

© Todos los derechos reservados

Bibliografía

[1] Ben-Ari, M. (2012). *Mathematical Logic for Computer Science*, DOI10.1007/978-1-4471-4129-7_3, © Springer-Verlag London.

[2] Bryant, R.E. (1986). *Graph-based algorithms for Boolean function manipulation*. IEEE Transactions on Computers, 8(C-35):677-691.

[3] Bryant, R.E. (1992). *Symbolic Boolean manipulation with ordered binary-decision diagrams*. ACM Computing Surveys, 24(3):293-318.

[4] Cardona Torres, S. A.; Hernández Rodríguez, L. A.; Jaramillo Valbuena, S (2010). *Lógica matemática para Ingeniería de Sistemas y Computación*. Editorial ELIZCOM S.A.S.

[5] Garrido, L.G. (2014). *Introducción a la Teoría de Conjuntos y a la Lógica*. Universidad de la Habana. Cuba

[6] Grimaldi, R.P. (1994). *Matemáticas Discreta y Combinatoria. Una introducción con aplicaciones* (3era ed.). Addison-Wesley Iberoamericana, ISBN 0-201-65376-1, Wilmington, Delaware, E.U.A.

[7] Hortalá González, M.T.; Leach Albert, J.; & Rodríguez Artalejo, M. (2001). *Matemática discreta y lógica matemática* (2da ed.). España: Complutense.

[8] Johnsonbaugh, R. (2005). *Matemáticas Discretas* (6ta ed.). México: Pearson Educación.

[9] Lira Contreras, A. R. (2005). *Lógica Elementos Teóricos*. Lugar: Ediciones Umbral.

[10] Martín Gonzalez, J.L. et al. (2007). *Problemas resueltos de electrónica digital* (1era ed.). Madrid, España: Delta Publicaciones.

[11] Paniagua Arís, E.; Sánchez González, J.L.; Martín Rubio, F. (2003). *Lógica computacional*. Editorial Paraninfo.

[12] Real Academia Española (2001). *Diccionario de la lengua española* (22da ed.). España: ESPASA.

[13] Rosen, K.H. (2012). *Discrete Mathematics and Its Applications* (7ma ed.). New York: McGraw-Hill.

[14] Raymond, S. (1978). *What Is the Name of This Book?*: *The Riddle uf Dral, ulu and Other Logical Puzzles*, Prentice-Hall, Englewood Cliffs, NI.

[15] Tocci, R.J.; Widmer, N. S. (2003). *Sistemas digitales: principios y aplicaciones*. (8va ed.). México: Pearson Educación.

© Todos los derechos reservados

© Todos los derechos reservados