LÓGICA DE PRIMER ORDEN

Mª Jesús Castel de Haro

Licenciada en Matemáticas

Faraón Llorens Largo

Diplomado en Profesorado de E.G.B.
Licenciado en Informática

Departamento de Ciencia de la Computación e Inteligencia Artificial
Universidad de Alicante

Mª Jesús Castel de Haro
Faraón Llorens Largo

Dpto. de *Ciencia de la Computación e Inteligencia Artificial*
Universidad de Alicante

Teléfono 96 590 39 00
Apartado de Correos, 99
03080 ALICANTE

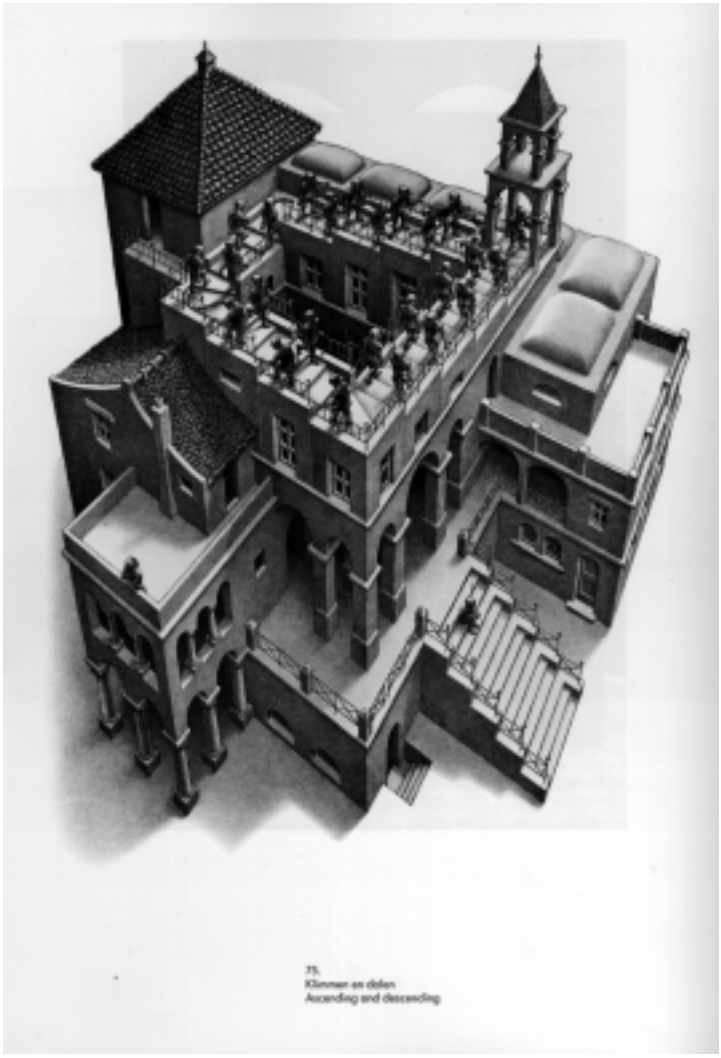
e-mail : [chus,faraon,logica]@dccia.ua.es

1ª Edición, Febrero 1996
2ª Edición, Marzo 1999

Título : LÓGICA DE PRIMER ORDEN
Autores : Mª Jesús Castel de Haro
Faraón Llorens Largo
Imprime : Ramón Torres Gosalvez
Teléfono (96) 590 34 00 Ext. 3390
Alicante
Impreso en España

I.S.B.N. : 84-922775-5-6

Depósito Legal : A-162-1999



75.
Klimmen en dalen
Ascending and descending

M. C. Escher

Contenido

Prólogo

Estos papeles que ahora tenéis en vuestras manos, nacieron con la idea de servir de material de apoyo a las clases de la asignatura “Lógica de Primer Orden”, que forma parte del programa de los estudios de Ingeniería Informática impartidos en la Escuela Politécnica Superior de la Universidad de Alicante.

No se trata, por tanto, de un libro de texto sobre Lógica, ya que existen excelentes libros que cubren este objetivo y es una pérdida de tiempo volverlos a escribir. En ningún momento ha sido esa nuestra pretensión, y si es eso lo que buscáis os remitimos a la bibliografía que adjuntamos. Lo que hemos intentado es redactar unos apuntes que incluyan los temas y aspectos de la lógica que nosotros impartimos en nuestras clases y que aborden la Lógica desde un enfoque didáctico y al mismo tiempo ameno (al menos es esa nuestra intención). Tampoco hemos dejado de lado el hecho de que nuestros alumnos y alumnas cursan estudios de Informática, y por ello hemos puesto hincapié en aquellos aspectos de la Lógica que creemos pueden ser interesantes para ellos como futuros informáticos, y a la vez les sirvan de complemento y base para otras materias de la carrera. Por tanto, hemos estado más interesados en mostrar la utilidad y aplicación de la lógica que en la formalización matemática de la misma, evitando las largas demostraciones de teoremas (para ello existen buenos libros en la bibliografía).

La Lógica de Primer Orden nos permitirá formalizar conocimientos que poseamos de diversas ramas del saber, convirtiéndose en un instrumento imprescindible para realizar un trabajo serio en muchas tareas científicas. En concreto esta formalización será un paso previo e indispensable para poder automatizar formas de razonamiento y su posterior aplicación a muchas de las áreas de la Informática, y fundamentalmente de la Inteligencia Artificial.

Esta reflexión general nos ha llevado a configurar el libro con la siguiente estructura :

Un primer capítulo introductorio que hace un breve paseo por la Historia de la Lógica. El segundo capítulo describe el Lenguaje de la Lógica de Primer Orden, y por

tanto el vocabulario que utilizaremos a partir de ese momento y que nos permitirá formalizar el conocimiento haciendo explícitos los objetos y las relaciones, así como sus restricciones. El capítulo tercero aborda el aspecto Semántico de la lógica, es decir, trata con el significado de nuestras sentencias. A diferencia de los capítulos cuarto y quinto que tratan el aspecto sintáctico de la lógica desde dos métodos de cálculo diferentes, uno utilizando la Deducción Natural y el otro la Teoría Axiomática. Los capítulos sexto y séptimo preparan el camino para llegar al final de la asignatura dando las bases de la Programación Lógica. El capítulo 6 explica los procedimientos para Normalizar las Fórmulas y el 7 sienta las Bases Teóricas para la Demostración Automática. Como apoyo, incluimos un anexo en el que describimos el lenguaje de programación lógica más extendido, el Prolog.

Ante todo, esperamos que estos apuntes os puedan servir de ayuda y os agradeceríamos cualquier sugerencia y comentario acerca de ellos. Ello redundaría en beneficio de futuros estudiantes.

Para finalizar este prólogo quisiéramos dejar constancia de nuestro agradecimiento al departamento de Tecnología Informática y Computación de la Universidad de Alicante, así como a los distintos alumnos y alumnas que han soportado estoicamente el suplicio de nuestras clases. También quisiéramos agradecer a nuestro compañero Juan Antonio Puchol su inestimable colaboración en la realización de la portada.

Alicante, enero de 1996

Mª Jesús y Faraón

Con la iniciación el curso 1997/98 de los estudios de Licenciatura en Matemáticas en la Universidad de Alicante y la impartición por parte de nuestro departamento de la asignatura de primer curso “Lógica de Primer Orden”, hemos revisado este material para que pueda adaptarse también a estos estudios. En concreto se ha desligado el Teorema de Deducción y las Propiedades Formales del tema 5 formando un nuevo tema que hemos llamado Metalógica. Al mismo tiempo se ha reforzado el tema de la Teoría Axiomática ya que los sistemas axiomáticos, en los cuales hay por un lado unos “axiomas lógicos” y por otro una serie de reglas de inferencia, son más formales y por ello pueden parecer más adecuados para los estudiantes de matemáticas. Ello sin ir en detrimento de los sistemas de razonamiento no axiomátizados, que partiendo de premisas y paso a paso van llegando de una forma más natural a las conclusiones utilizando únicamente reglas de inferencia, vistos en el tema de Deducción Natural.

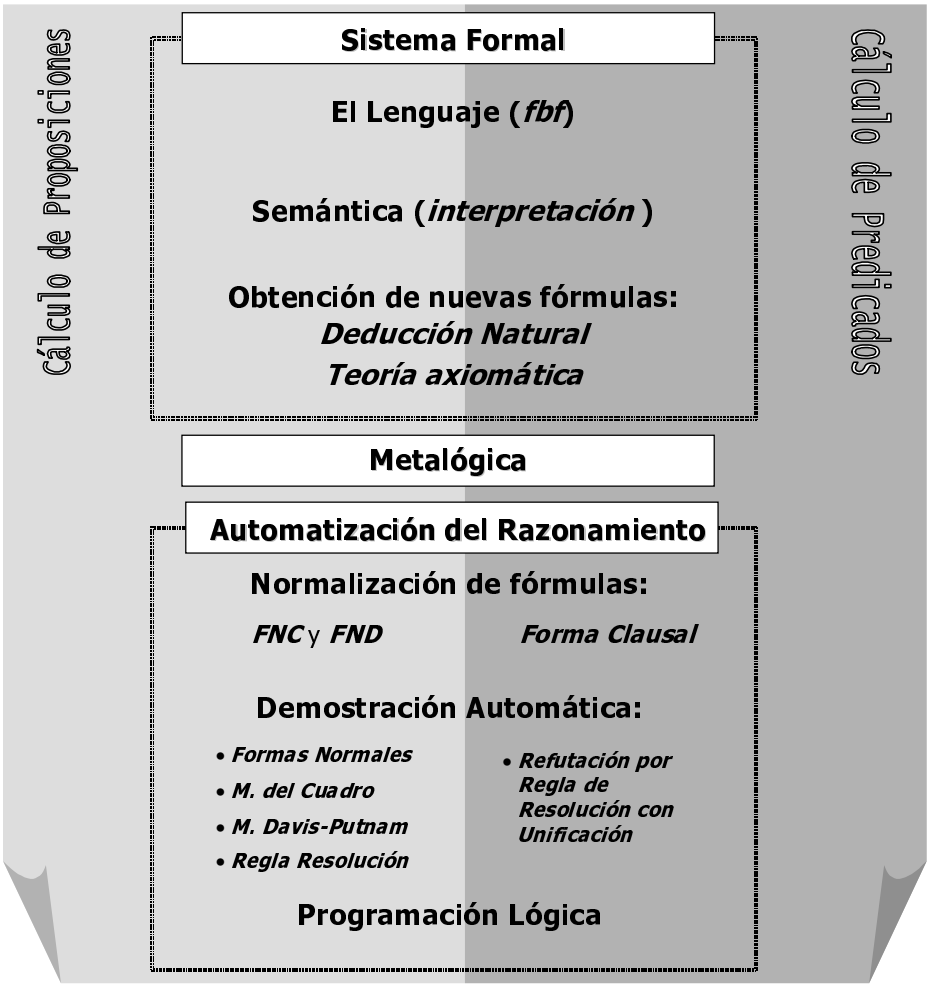
En esta segunda edición se ha corregido los errores detectados en la edición anterior así como se han ampliado y actualizado las referencias bibliográficas. Para cualquier sugerencia se dispone de una dirección de correo electrónico a la que os podéis dirigir :

logica@dccia.ua.es

Por último hemos trazado un “mapa” del libro que puede ayudar a adquirir una visión global del mismo:

Alicante, septiembre de 1998

Mª Jesús y Faraón


Cálculo de Proposiciones
Cálculo de Predicados
Sistema Formal
El Lenguaje (fbf)
Semántica (interpretación)
Obtención de nuevas fórmulas:
Deducción Natural
Teoría axiomática
Metalógica
Automatización del Razonamiento
Normalización de fórmulas:
FNC y FND
Forma Clausal
Demostración Automática:
Formas Normales
M. del Cuadro
M. Davis-Putnam
Regla Resolución
Refutación por Regla de Resolución con Unificación
Programación Lógica

Capítulo **1**

HISTORIA DE LA LÓGICA SIMBÓLICA

Se trata de un tema introductorio basado en la máxima "Sólo conociendo el pasado se puede entender el presente". Si se conoce cual ha sido la evolución de una ciencia y el contexto histórico en el que se ha desarrollado, se comprenderá mejor lo que se sabe de ella actualmente permitiéndonos afrontar de una manera más adecuada el presente y, por ello, se evitará, en parte, caer en los mismos errores del pasado. No olvidemos que los conocimientos del momento surgieron como respuesta a problemas que se plantearon en su tiempo. En este capítulo daremos un rápido repaso por la historia de la Lógica, desde Aristóteles hasta nuestros días, en los que la revolución de la Informática ha representado un nuevo auge de la lógica. También veremos, de manera muy breve, en que consiste la Lógica de Primer Orden.

El contexto de este libro se desarrolla en el entorno de la Lógica Matemática, Lógica Simbólica, Lógica Formal o simplemente Lógica. Hay autores que distinguen estos tipos de Lógicas, diciendo que la Lógica Formal es un saber que abarcaría no sólo a la Lógica Matemática, sino también a la Lógica Tradicional, aquella que fue desarrollada desde Aristóteles hasta Leibniz. Se llama Lógica **Formal** porque su interés es la forma a los enunciados, razonamientos,..., sin tener en cuenta su contenido; también decimos Lógica **Matemática** porque utiliza símbolos como los matemáticos; y **Simbólica** porque se trata de una escritura con símbolos.

La Lógica es la disciplina que contiene como verdades aquellas en las que sólo figuran esencialmente ocurrencias de ciertas expresiones, que son de uso general en todos los saberes. La Lógica se creó como ciencia dedicada a la identificación de las formas humanas del razonamiento, con el objetivo de crear criterios para discernir la corrección o no de los argumentos usados en discusiones filosóficas de los antiguos

griegos; hoy día es un instrumento fundamental en la construcción de ordenadores, en la creación de lenguajes de programación (tales como Prolog y Lisp) y en los sistemas expertos de Inteligencia Artificial. Dentro de otros campos, la Lógica Simbólica es también un instrumento imprescindible para el ejercicio de una tarea seria tanto en Ciencia como en Filosofía. Sus aplicaciones se dan en campos tan dispares como las Matemáticas, la Lingüística, las Ciencias Naturales y Sociales, la Jurisprudencia y la Filosofía, entre otros. Y podemos pensar ¿qué tienen en común disciplinas tan diversas para que todas utilicen la lógica? Evidentemente ni la materia de que tratan ni en la metodología que utilizan. Lo que estos, y otros campos del saber, tienen en común es la dependencia de una cierta «racionalidad»: utilizan argumentaciones racionales basadas en principios aceptados.

Como dice Frege en su trabajo "El pensamiento: una investigación lógica" [1] :

> «Así como la palabra *bello* señala la dirección de la estética y *bueno* la de la ética, del mismo modo *verdadero* señala la de la lógica. Todas las ciencias tienen la verdad como meta, pero la lógica se ocupa de ella de una manera completamente diferente /... / a la lógica le toca decretar las leyes del ser verdad»

Cuando se nos plantea un nuevo tipo de problemas que debemos resolver, bien manualmente o por medio del ordenador, debemos empezar siempre planteándonos una serie de preguntas acerca del *conocimiento* implicado en dicho problema: ¿qué tipo de conocimiento precisa? ¿cómo debemos representar ese conocimiento? ¿cuánto conocimiento es necesario? ¿cuál es el conocimiento requerido? ¿cómo derivar nuevo conocimiento a partir del que tenemos? ... Y en la representación y manipulación de ese conocimiento es donde la Lógica puede jugar un papel importante.

La palabra *lógica* se utiliza profusamente en el uso común y cotidiano de la lengua. Así, casi todos hemos dicho alguna vez "lógicamente" o "es lógico". Pero ¿qué entendemos por lógica? Si consultamos un diccionario[2] para ver el significado del término lógica encontraremos:

> «**lógica**. f. Ciencia que expone las leyes, modos y formas del conocimiento científico»

Si queremos una definición más completa podemos acudir a una enciclopedia[3], donde podemos hallar algo parecido a:

[1] "Der Gedanke. Eine logische Untersuchung" publicado en *Beiträge zur Philosophie des deutschen Idealismus*, 1, 1918-19, pág. 58-77.

[2] *Diccionario de la Lengua Española*. Real Academia Española, Madrid 1992

[3] *Salvat Universal*. Diccionario Enciclopedico, Tomo 13

«**lógica**. (Del lat. *logica*, y éste del gr. *logiké*, t. f. de *logikós*, lógico) f. Ciencia que estudia las condiciones formales de validez de una inferencia y, en general, de una argumentación cualquiera»

Veamos ahora una definición de lógica, sencilla pero que nos servirá a nosotros como punto de partida:

Lógica : disciplina que estudia la estructura, fundamento y uso de las expresiones del conocimiento humano.

Podemos considerar la lógica como una teoría de la forma o estructura del razonamiento deductivo, que prescinde del contenido y tiene como tarea primordial el estudio de la Lógica ordinaria con métodos matemáticos, donde la lógica ordinaria se compone del lenguaje ordinario y de una serie de frases de dicho lenguaje, que se llaman deducciones. También podemos decir que la lógica, en su vertiente matemática, está relacionada con los criterios de validación de inferencias y los principios formales del razonamiento.

Así, el lenguaje de la lógica se convierte en uno de los mecanismos de representación del conocimiento en Inteligencia Artificial, con la ventaja de que además proporciona un método, la deducción matemática, para obtener nuevo conocimiento a partir del antiguo.

1. ETAPAS DE DESARROLLO HISTÓRICO DE LA LÓGICA

El estudio de la historia de una determinada ciencia nos proporcionará una perspectiva global de la misma y su estructura general, así como la relación existente con otros campos del saber. En cuanto a la lógica, podríamos agrupar su evolución en tres etapas: una primera que arrancaría con sus inicios en la antigüedad clásica, una segunda caracterizada por su acercamiento a las matemáticas y la última, en el presente siglo, emparejada con el rápido avance de la informática.

1.- Aplicación a la Filosofía

La Lógica Formal nació hace 2.500 años, cuando Aristóteles y los Estoicos se interesaron por la construcción y el análisis de esquemas de argumentos. Desde entonces no ha experimentado grandes cambios hasta el siglo XIX en que se propuso su matematización. Durante casi dos mil años la codificación de Aristóteles de las formas válidas de deducción fue universalmente considerada como completa e incapaz de mejora esencial. En 1.787 el filósofo alemán E. Kant pudo decir que desde Aristóteles la

lógica formal "no ha sido capaz de avanzar un solo paso, y según todas las apariencias, es un cuerpo de doctrina cerrado y completo". A la Lógica Formal, tal como venía siendo desde Aristóteles hasta Kant, se le ha llamado Lógica Tradicional. A la Lógica actual matematizada se le llama Lógica Simbólica o Matemática.

2.- Matematización de la Lógica

Por matematizar la Lógica se entiende, en metodología científica, la subordinación de una ciencia al método de la matemática, es decir, extender el método matemático a la Lógica. La matematización de la Lógica la entendemos en la medida en que ésta incorpora a sus técnicas de trabajo la *exactitud* y *rigor* matemático. Para ello es necesario la definición de un lenguaje formal y la utilización de unas reglas operacionales precisas.

El fundador de la Lógica Simbólica es el pensador G.W. Leibniz (siglo XVIII) que concibió la idea de aplicar las técnicas de deducción matemática a los razonamientos filosóficos. A este fin propuso construir un cálculo ideológico «*Calculus ratiocinator*» que consistía en un sistema de reglas que permitía operar con las ideas de un modo exacto, parecido a las matemáticas. Para ello tenía que construir un lenguaje artificial, que fuera perfecto y que no contuviera ningún tipo de ambigüedad. Leibniz no llegó a realizar este proyecto. Pero dos ilustres matemáticos del siglo XIX como George Boole y Gottlob Frege tomaron estas ideas.

Boole publicó en 1854 la obra "Laws of Thought" (Las Leyes del Pensamiento)[4] en donde independientemente de Leibniz descubrió y formuló las leyes de un álgebra del pensamiento o *Álgebra Lógica*, que tienen el mismo rigor que las leyes del álgebra matemática.

Frege publicó en 1879 una obra titulada "Begriffsschrift" (Conceptografía)[5] en la que consigue la construcción de un cálculo Lógico, con una escritura artificial perfecta que permite la formalización de la Lógica deductiva elemental.

Por ello, Boole y Frege son los grandes creadores de la Lógica Simbólica o Matemática. Podemos citar otras figuras del siglo XIX importantes por sus trabajos en Lógica como Bolzano, Peirce, Morgan, Venn y Jevons, entre otros muchos.

Ya en el siglo XX destaca la escritura simbólica de Peano y la obra "Principia Mathematica" de Whitehead y Bertrand Russell. En la década de los años 20 tiene gran

[4] George Boole, "An Investigation of the Law of Thought, on which are founded the Mathematical Theories of Logic and Probabilities", 1854. Publicada por Editorial Paraninfo en 1982 "Una investigación sobre las leyes del pensamiento".

[5] Gottlob Frege, "Begriffsschrift: eine der arithmetischen nachgebildete Formalsprache des reinen Denkens", 1879 (Conceptografía : un lenguaje simbólico al modo aritmético del pensamiento como tal)

importancia la obra de Brouwer y Hilbert, creadores respectivamente de la escuela intuicionista y de la escuela formalista, que se mueven en problemas de la fundamentación de las matemáticas. También sobresalen autores como Lukasiewicz y Wittgenstein. Ya en la década de los 30 podemos destacar:

- Descubrimiento por Gentzen de los cálculos de deducción natural.
- Establecimiento de los teoremas de limitación de Kurt Gödel y Church, y de las teorías de computación de Emil Post y Alan Turing.
- Desarrollo de la semántica de Alfred Tarski.

En la 2ª mitad del siglo XX la Lógica cubre muchas áreas. Entre las principales aplicaciones a partir de los años 50 destaca la formulación del método de las tablas semánticas de Beth, y los hallazgos metalógicos de Henkin y Craig.

3.- Extensión a la tecnología a través de la Informática

La década de los 60 fundamentó dos técnicas de apoyo de la Lógica como son:

- Búsqueda heurística.
- Deducción automática.

Esta última se basaba en la utilización de teorías lógicas disponibles de forma que pudieran formularse problemas dentro del límite de indecibilidad que tiene la Lógica de Primer Orden, con lo cual se produjeron teorías como la regla Universal de resolución con unificación (Robinson, 1965). A partir de aquí se puso de relieve por Green la forma de extracción de respuestas a problemas a partir de resolución y unificación. Colmerauer (1972) hizo la 1ª revisión de Prolog, y se dio un impulso en I.A. para sistemas expertos como modelos computables del conocimiento no totalmente sistematizado.

A primeros de los 80 la Lógica constituye una base en el desarrollo de las técnicas informáticas, enfocadas principalmente en tres grandes líneas:

- Lógica de Hoare como soporte de las técnicas de *verificación de programas* procedurales.

- Nuevo enfoque de los lenguajes de programación: *Programación Lógica*. Estos son capaces de formular tanto los algoritmos de programación clásicos como las bases de conocimientos para sistemas expertos.

- Modelización del *razonamiento de "sentido común"* para los sistemas expertos, tanto en Lógica Clásica, Difusa ("Fuzzy Logic") o no monótona.

Así, llegamos a los ordenadores de 5ª generación cuya idea fundamental es producir máquinas adaptadas tanto a procesar *datos* como *conocimientos* por lo que se plantean basados en una instrumentación hardware de un lenguaje de programación

Lógico (su lenguaje máquina será un derivado del Prolog). Parece ser que este proyecto no ha llegado a ver la luz.

La Lógica Formal ha jugado un importante papel en la Inteligencia Artificial en los últimos años. Podemos agrupar en tres los grandes campos de aplicación de la lógica a la I.A. :

- la lógica como herramienta analítica.
- la lógica como representación del conocimiento y sistema de razonamiento: usar la lógica como un formalismo para la representación del conocimiento en los sistemas de ordenadores inteligentes y aplicar la deducción lógica para proyectar inferencias desde el conocimiento así representado.
- la lógica como lenguaje de programación.

Pero quizás, el hecho más destacable del estado actual de la lógica en general ha sido el desarrollo de las llamadas **lógicas no clásicas**, surgidas, en parte, como respuesta a la aplicación de la lógica a la Inteligencia Artificial. Se trata de sistemas lógicos que difieren en una o más cualidades de la lógica clásica. Las características que asumimos en los sistemas lógicos clásicos serían, entre otras:

– los enunciados están provistos de un valor de verdad	***apofántica***
– dichos valores de verdad son únicamente dos: *verdadero* o *falso*	***bivalente***
– no existen matizaciones entre estos valores	***asertórica***
– las conexiones entre enunciados dan lugar a enunciados compuestos cuyo valor de verdad está completamente en función de los valores de verdad de los enunciados conectados	***extensional***

Con respecto a estas características, y como extensiones a la lógica “normal”, van surgiendo distintas lógicas “desviadas” o “divergentes”:

Polivalente [6]	Lukasiewicz[7], Post[8], Rosser[9]	Cuestionan el principio de bivalencia. Tenemos lógicas finitamente polivalentes (trivalente,...) e infinitamente polivalentes.

[6] N. Rescher, “Many-Valued Logic”, Nueva York, McGraw-Hill, 1969

[7] J. Lukasiewicz, “O logice trójwartosciowej”, en *Ruch Filozoficzny*, 5, 1920, pág. 170-171. “Sobre la lógica trivalente”, ver. castellano J. Lukasiewicz, “Estudios de lógica y filosofía”, selección, traducción y presentación de A. Deaño, Madrid, Revista de Occidente, 1975, pág, 41-42

[8] E. Post, “Introduction to a General Theory of Elementary Propositions”, *American Journal of Mathematics*, vol. 43, 1921, pág. 163 y siguientes. Reimpresión en J. Van Heijenont (ed.) “From

Difusa ("*fuzzy*")	Zadeh[10]	Se trata de una lógica polivalente que surge por necesidad, no por principio, y que trata el tema de la vaguedad (ambigüedad, imprecisión, carácter borroso de ciertos términos,...).
Probabilística	Bacchus[11], Pearl[12]	Aplicación de las probabilidades y el conocimiento estadístico a sistemas formales de razonamiento y representación del conocimiento.
Modal [13]	Lewis, Carnap[14], Kripke	Incorpora matices a la valoración de verdad de los enunciados, admitiendo modalidades de esa verdad: necesario, posible, imposible,...
Deóntica	Wright[15]	Se puede considerar una rama de la Lógica Modal. Se ocuparía de las relaciones de inferencia entre normas, es decir, entre proposiciones prescriptivas (obligatorias).
No monótonas	McDermott[16], McCarthy[17]	La obtención de nueva información puede llevarnos a revisar y/o cambiar creencias anteriores.

Frege to Gödel", Cambridge, Mass., Harvard University Press, 1967, pág. 264 y siguientes. Ver. castellano A. Deaño (ed.) "Lecturas de Lógica Formal". Madrid, Alianza Editorial

[9] J. B. Rosser y A. R. Turquette, "Many-Valued Logic", Amsterdam, North Holland Pub. Co. 1952

[10] L. Zadeh, "Fuzzy Sets", *Information and Control*, vol. 8, 1965, pág. 338-353

[11] F. Bacchus, "Representing and Reasoning with Probabilístic Knowledge: A Logical Approach to Probabilities", The MIT Press, Cambridge,Massachusetts, 1990

[12] J. Pearl, "Probabilistic Reasoning in Intelligent Systems", Morgan Kaufmann, San Mateo, California, 1988

[13] G. E. Hughes y M. J. Cresswell, "An Introduction to Modal Logic", Londres, Methuen and Co. 1968. Ver. castellana : "Introducción a la lógica modal", Madrid, Tecnos, 1973

[14] R. Carnap, "Meaning and Necessity (A Study in Semantics and Modal Logic)", Chicago y Londres, The University of Chicago Press, 1947, 2ª ed. ampliada 1956

[15] G. H. V. Wright, "Deontic Logic", *Mind*, vol. LX, 1951, pág. 1-15

[16] D. McDermott y Jon Doyle, "Non-Monotonic Logic I", *Artificial Intelligence*, 13, 1980, pág. 41-72

[17] J. McCarthy, "Programs with Common Sense", en *Mechanization of Thought Processes*, Proceedings of the Symposium of the National Physics Laboratory, Londres, 1958, pág. 77-84. Reimpresión en *Semantic Information Processing*, MIT Press, Cambridge, MA, 1968, pág. 403-418

Temporal	Gardies[18], Allen, McDermott	Reconoce la existencia de esquemas de inferencia específicamente temporales, donde una misma sentencia puede tener diferente valor de verdad en diferentes momentos.
Intuicionista	Heyting[19], Dummett[20]	...
Combinatoria	Curry[21]	...
Epistémica	Hintikka	...
...	...	...

2. LÓGICA DE PRIMER ORDEN

El libro desarrolla la *Lógica de Primer Orden*, en la que se trabaja a dos niveles de complejidad con los siguientes formalismos:

- *Cálculo de Proposiciones*, de *Enunciados* o de *Conectores*: el cual está destinado al estudio lógico de los enunciados tomados en bloque (o sea sin analizar en sus partes) considerando únicamente las relaciones o combinaciones externas (conexiones de carácter lógico) de unos con otros. Sus símbolos van a ser variables proposicionales (p, q,...) y conectivas.

- *Cálculo de Predicados* o *Cuantificacional*: que analiza los enunciados en el caso más sencillo, es decir, compuesto por objetos y relaciones. La interpretación la realiza mediante variables y constantes para los objetos, predicados, conectivas y cuantificadores.

[18] J. L. Gardies, "La logique du temps", Paris, PUF, 1975

[19] A. Heyting, "Intuitionism. An Introduction", Amsterdam, North-Holland Publishing Co. 1956, 3ª ed. 1971. "Introducción al Intuicionismo", Madrid, Ed. Tecnos, 1976

[20] M. Dummett, "Truth", Proceedings of the Aristotelican Society, vol. 59, 1958-59, pág 141 y siguientes. Reimpresión : P.F. Strawson (ed.) "Philosophical Logic", Oxford University Press, 1967, pág. 49 y siguientes

[21] H. B. Curry y R. Feys, "Combinatory Logic", Amsterdam, North-Holland Pub. Co. 1958. Ver. castellana Madrid, Editorial Tecnos, 1967, pág. 19

La ventaja de la Lógica Proposicional es su simplicidad y la disponibilidad de un proceso de decisión. De ahí que a lo largo del libro, la mayoría de los capítulos empiecen trabajando inicialmente en el cálculo de proposiciones para, una vez comprendido y asimilado, pasar a ampliar y generalizar los conceptos para el cálculo de Predicados.

La lógica de primer orden (entendiéndola a partir de ahora como sinónimo de cálculo de predicados) tiene como tarea primordial el estudio de la Lógica ordinaria con métodos matemáticos. La lógica clásica fue creada para razonar sobre los principios matemáticos, donde la ambigüedad y la imprecisión no son bien vistos. Cuando queremos aplicar la lógica a ejemplos no matemáticos, primero debemos "matematizarlos", es decir crear un modelo matemático que represente de manera ideal a ese mundo. Ahora, la lógica clásica ya puede ser usada para razonar correctamente sobre ese modelo. Así, en la lógica clásica investigamos los principios del razonamiento para mundos perfectos.

La Lógica ordinaria se compone del lenguaje ordinario y de una serie de frases básicas de dicho lenguaje, que se llaman ENUNCIADOS. Un conjunto de dichas frases va a ser lo que llamaremos ARGUMENTO o DEDUCCIÓN. Para trabajar, vamos a utilizar símbolos de VARIABLES. Al igual que en matemáticas, las variables van a indicar los lugares que pueden ser ocupados por nombres de cosas, que en nuestro caso serán enunciados en el cálculo de proposiciones y objetos en el de predicados. Estos enunciados, que estarán sin determinar, van a tener un valor que diremos VERDADERO o FALSO, y con ellos construiremos un SISTEMA DE CÁLCULO.

Esta significación de los sistemas Lógicos como cálculos interpretados procede de Carnap, el cual indica que un enunciado cualquiera se interpreta por una variable (p, q,...) susceptible de interpretación por su valor de verdad (V ó F). Con dichas variables construiremos FÓRMULAS y veremos que cuando una fórmula es verdadera para toda interpretación posible en el Universo de Discurso relevante, esa fórmula es un enunciado formalmente válido.

Hablamos de **Lógica de Primer Orden** porque los cuantificadores se aplican exclusivamente a las variables que representan términos (objetos). Podríamos cuantificar variables de funciones o, incluso, predicados; a este cálculo de predicados se le denomina de orden superior. El trabajo con estructuras deductivas en cálculos de orden superior quedan fuera del ámbito de este libro y por tanto no entraremos a tratarlo.

Para cada formalismo lógico (Cálculo de Proposiciones y Cálculo de Predicados) vamos a estudiar dos aspectos: la **Teoría Sintáctica** y la **Teoría Semántica**.

Sintaxis : estudia en un Lenguaje o en un Sistema Formal, las relaciones de

unos signos y unas fórmulas con otros signos y otras fórmulas (*fórmulas bien formadas*). Realiza una exposición del lenguaje formal (tablas de símbolos y reglas de formación de fórmulas) y cálculo fundado en la relación formal de deducibilidad (teoría de la deducción formal). Son categorías sintácticas conceptos como *forma lógica*, *fórmula*, *fórmula bien formada*, *deductor*, etc. Dentro de la teoría sintáctica y para cada formalismo lógico veremos:

- **Deducción Natural** (reglas de inferencia).
- **Tª de la Demostración** (axiomas y teoremas).

Semántica : relación de los signos y fórmulas con sus contenidos y objetos extralingüísticos. La relación entre fórmulas no se construye a partir de axiomas y reglas de inferencia sino mediante una simbolización del significado de las proposiciones, es decir, de la forma de valorar el contenido de cada proposición. Categorías semánticas son, por ejemplo, las de *valores de verdad*, *interpretación*, *modelos*, *verdad*, *satisfacibilidad*, ...

3. TRABAJOS COMPLEMENTARIOS

1.- Descripción y fundamentos básicos de "otras" lógicas (lógicas no clásicas) :

- Lógica Temporal
- Lógica Modal
- Lógica Multivalente
- ...

2.- Estudio y comentario del artículo : "The Role of Logic in Artificial Intelligence", incluido en el libro *LOGIC and REPRESENTATION* de Robert C. Moore, CSLI Lecture Notes nº 39, Center for the Study of Language and Information, Stanford, California, 1995 [22]

3.- El problema de la representación del conocimiento. Introducción a los distintos métodos y técnicas[23]

[22] Artículo originariamente aparecido bajo el nombre "The Role of Logic in Intelligent Systems" en *Intelligent Machinery : Theory and Practice.* Ed. I. Benson, Cambridge, England : Cambridge University Press, 1986

[23] Se puede encontrar información al respecto en el libro "Inteligencia Artificial" de Elaine Rich y Kevin Knight, McGraw-Hill, 1994

4. LA LÓGICA EN LA VIDA

Ludwing Wittgenstein, filósofo austriaco, apuntó que "podría escribirse una obra filosófica buena y seria compuesta enteramente por chistes". Si se entiende el chiste, se entenderá el argumento implícito. En esta sección vamos a contar chistes, historias, juegos y acertijos, relacionados de alguna manera con la Lógica. La enseñanza de la lógica no tiene porque ser aburrida si se saben encontrar caminos para hacerla agradable y distraída. Como dice Martin Gardner[24] la virtud (y creemos que lo difícil como siempre en esta vida) está en encontrar el equilibrio entre el juego y la seriedad: el juego hará que los alumnos y alumnas se muestren interesados en la actividad y sigan hablando de ella fuera de las clases; la seriedad convertirá las clases en algo útil y provechoso (y, para algunos, justificará nuestro sueldo).

1. *¿Qué es lógica?*

"- Ya sé lo que estás pensando - dijo Tweedledum -; pero no es como tú crees. ¡De ninguna manera!

- ¡Por el contrario! - continuó Tweedledee -. Si fue así, entonces podría ser; y si fuera así, sería; pero como no es, no es. ¡Eso es lógica!"

Lewis Carroll, "Alicia a través del espejo"

2. *¿Tiene lógica?*

Una mujer se burla de su marido enloquecido, que tiene un revólver cargado apoyado en la sien.

- No te rías - dice él -. Después vas tú.

3. *¡Vaya lógica!*

"Ruegoles acepten mi renuncia. No deseo pertenecer a ningún club que me acepte como miembro"

Groucho Marx, "Groucho y Yo"

4. *¡Lógica aplastante!*

"Hay gente que quiere conseguir la inmortalidad mediante sus obras o las de sus descendientes. Yo quiero conseguir la inmortalidad no muriéndome"

Woody Allen

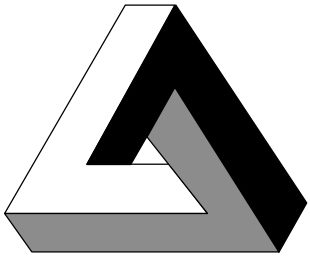
[24] Martin Gardner, "Carnaval Matemático", Alianza Editorial (libro de bolsillo 778), 1.987

5. *¡Es lógico!*

Dos amigos están en un restaurante y ambos piden pescado. El camarero les trae una fuente con dos trozos. Uno de elloseducadamente dice: “Sírvete, por favor”. El otro se sirve el trozo más grande. Tras una inquisitiva mirada, el primero le reprocha: “Si yo me hubiese servido antes, me habría puesto el trozo más pequeño”. A lo que el segundo contesta: “De que te quejas, es el que te he dejado”.

5. *¡Sin lógica!*

Junto al mundo real, lógico, aceptado por la mente racional, existe un mundo irreal o imposible, de cosas que nunca existieron, pero que podemos pensarlas e imaginarlas. Tal como dice Escher[25] una etapa en la elaboración de una estampa es “la búsqueda de una forma visual capaz de traducir del modo más claro posible un determinado pensamiento. /.../ Un pensamiento, sin embargo, es algo enteramente distinto a una imagen visual”. Una figura imposible es un objeto representado en perspectiva en un plano, que es imposible de ser construido en el espacio, como la escalera de la portada interior “Ascending and descending” que nos permite ascender/descender infinitamente, o la siguiente figura triangular :



[25] M.C. Escher, *Estampas y dibujos*, página 5, Benedikt Taschen Germany 1994

Capítulo **2**

EL LENGUAJE DE LA LÓGICA DE PRIMER ORDEN

La lógica pretende formalizar las expresiones del conocimiento humano. Y dicho conocimiento lo adquirimos y transmitimos por medio del lenguaje. Pero el lenguaje natural que utilizamos es ambiguo y engorroso, y por ello, para trabajar con el conocimiento formalmente, necesitamos un lenguaje artificial. En un primer momento trabajaremos con proposiciones, es decir, frases declarativas simples tomadas en bloque. En una segunda parte descompondremos las frases en objetos y relaciones/propiedades, dando lugar a los conceptos de términos y predicados. Al final del tema deberemos estar familiarizados con el Lenguaje de la Lógica de Primer Orden, esto es, las fórmulas bien formadas del Cálculo de Predicados, ya que a partir de este momento éste será nuestro medio de expresión de conocimiento.

Todos los seres vivos se comunican entre sí por medio del lenguaje natural correspondiente a cada especie. Es una de las vías por la cual se transmite el conocimiento, el cual está formado tanto por *hechos* o *ideas* (frases de tipo declarativo, interrogativo e imperativo), como por lo que se llaman *deducciones*, que son la consecuencia de un número determinado de declaraciones de las cuales se sigue una conclusión. Las oraciones declarativas constituyen el elemento básico para la descripción del conocimiento.

Ejemplos:
La ciencia es la estética de la inteligencia
El saber no ocupa lugar
Un cuadrado tiene los cuatro lados iguales y los ángulos rectos
Los polígonos irregulares tienen los lados o los ángulos desiguales
A quien Dios se las dé, San Pedro se las bendiga
Dos números son opuestos si y sólo si tienen el mismo valor absoluto y distinto signo

En martes, ni te cases ni te embarques
Sólo los informáticos entienden a los informáticos
Nadie entraba en la academia de Platón a menos que supiera Geometría

El lenguaje del cual partimos es el lenguaje natural del ser humano (en nuestro caso castellano), el cual consta de reglas sintácticas y semánticas, que nos permiten construir frases y razonamientos de forma correcta. Pero para desarrollar nuestro cálculo deberemos definir una lenguaje específico que nos permita trabajar de forma consistente. Por eso apoyándonos en el lenguaje natural vamos a definir un lenguaje artificial que tendrá unas bases concretas que se aplicarán al natural, un lenguaje perfectamente adaptado a su propósito, conciso y preciso, con reglas que no sufren excepciones. Como dice Polya[1]:

> «la elección de una notación constituye una etapa importante en la solución de un problema. Debe elegirse con cuidado. /.../ Una notación apropiada podrá contribuir de modo primordial a la comprensión del problema»

Pasamos a continuación a describir los objetos con los que vamos a trabajar en lógica indicando su formalización; más tarde nos ocuparemos en relacionarlos y en trabajar con ellos.

Argumento, Deducción o Estructura Deductiva : Es un segmento lingüístico de cierta complejidad en el cual, de la posición de subsegmentos iniciales se sigue necesariamente la posición de un subsegmento final.

Ejemplo:
Si estudias la lección, iremos al cine; has estudiado la lección, luego vamos al cine

Como vemos, un argumento es un trozo de lenguaje y es susceptible de análisis objetivo desde un punto de vista matemático. Las partes principales de que consta un argumento se llaman *"enunciados"*. Los enunciados iniciales de un argumento se llaman **premisas** y el enunciado final **conclusión.**

Enunciado : Segmento lingüístico que tiene un sentido completo y que puede ser afirmado con verdad o falsedad.

Ejemplos:
Mañana es sábado
Todo mamífero es vertebrado
Madrid es la capital de España

Expresión : Cualquier segmento de un enunciado (verbal o escrito) que es, o bien *autosemántico* (tiene significado en sí mismo, es decir. que designa un ente, cualquiera que éste sea, lingüístico o extralingüístico), o bien *sinsemántico* (posee un papel semántico determinado, consistente en que su colocación en ciertas posiciones

[1] *Cómo plantear y resolver problemas*, G. Polya, Ed. Trillas, México, 19ª edición, 1995. Título original "How to solve it", 1945.

junto a signos que por sí solos designan algún ente da por resultado el surgimiento de un signo complejo que designa también algún ente).

Ejemplo: Felipe II es ambicioso

Felipe II : signo autosemántico que designa al propio Felipe II.
ambicioso : signo autosemántico que designa la propiedad de ser ambicioso.
es : signo sinsemántico, su papel es su colocación entre *Felipe II* y *ambicioso* para dar por resultado otro signo que se llama ORACIÓN o ENUNCIADO y que designa algo.

Ocurrencia de una expresión en otra es la presencia de la 1ª en la 2ª como parte suya.

Ejemplo:
Hay una ocurrencia de la expresión «cielo» y otra de la expresión «raso» en la oración «cielo raso»

El argumento proviene de un lenguaje natural. Y este lenguaje no es exactamente el lenguaje que interesa a la Lógica, ya que depende de muchos factores. El lenguaje que nos interesa es un lenguaje artificial, que cuente con reglas explícitas por las que se establezca el uso de términos y la formación de enunciados. Otra característica interesante de este lenguaje artificial es su universalidad, ya que no queda limitado a un determinado idioma, y por tanto, a un número limitado de usuarios. El lenguaje al cual se sujeta la Lógica Formal es el llamado **Lenguaje Formal** o **Simbólico**: un lenguaje que contiene símbolos constantes y símbolos variables.

Lenguaje Formal : Conjunto de símbolos, *alfabeto*, más un conjunto de sucesiones finitas de esos símbolos, llamadas *palabras*, *frases* o *fórmulas bien formadas*.

Según la interpretación de lenguaje formal que utilizamos podemos definir dos formalismos, dentro de la Lógica de Primer Orden. Estos dos lenguajes son:

- **Lenguaje de Proposiciones**: Representación del lenguaje usual tomando como elemento básico de la formulación una representación matemática de las frases declarativas simples (*proposiciones*).

- **Lenguaje de Predicados**: Representación del lenguaje usual tomando como base los componentes de algunos tipos de proposiciones: *términos* y *predicados*.

A continuación, en las dos partes de que consta este tema, pasamos a exponer ambos formalismos: veremos la definición de su lenguaje, su fomalización y la forma de relacionarse sus símbolos. Antes, y para finalizar esta introducción, daremos una última definición.

Sistema Formal : Es un modelo matemático con tres componentes:

1.- *Lenguaje Formal* : Conjunto de fórmulas sintácticamente correctas.

2.- *Noción de Verdad*: Interpretación de esas fórmulas.

3.- *Método de Cálculo*: Deducción de fórmulas nuevas a partir de otras anteriores, mediante manipulación sintáctica, sin tener en cuenta el significado de los símbolos. Para ello el método dispone de:
   a) Conjunto de Axiomas.
   b) Reglas de Inferencia.
   c) Concepto de Deducción o Demostración.

Los puntos 1 y 3 constituyen el aspecto sintáctico de la lógica. El punto 2 el aspecto semántico. Nuestro objetivo, a partir de ahora, será definir un sistema formal para la Lógica, de forma que en los temas posteriores desarrollaremos este modelo matemático. Así, este primer tema está dedicado a estudiar el Lenguaje Formal de la Lógica de Primer Orden. El tema 2 tratará el aspecto Semántico (valoración de verdad). Y en los temas tercero y cuarto desarrollaremos dos métodos distintos de cálculo: la Deducción Natural y el Sistema Axiomático.

1. EL LENGUAJE DEL CÁLCULO DE PROPOSICIONES

A nivel intuitivo diremos que, en este nivel de simbolización, se considera el lenguaje constituido por:

- *Enunciados simples* o *proposiciones atómicas.*
- *Conectivas.*
- *Paréntesis.*

De una manera más formal diríamos que el Lenguaje del Cálculo de Proposiciones utiliza el siguiente:

Alfabeto : conjunto no vacío de símbolos que tiene las siguientes categorías:

- *variables proposicionales*: p, q, r, ...
- *conectivas lógicas*: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- *símbolos auxiliares*: (,), [,],

Enunciado simple o Proposición atómica : Unidad mínima de lenguaje (no se puede dividir) con un contenido de información (tengan sentido lógico), sobre cuyo significado es posible pronunciarse (podemos decir que son ciertas o falsas).

Veamos algunos tipos de enunciados simples:

- Enunciados de acción con sujeto no determinado:
  Llueve
  Hace frío
  Es invierno

- Enunciados de atribución de propiedades a sujetos determinados:
  Juan es informático
  Pepe es estudioso

  Sujetos: Juan, Pepe
  Propiedades: es informático, es estudioso

- Enunciados de relación entre sujetos:
  Juan es amigo de Pepe
  Zaragoza está a medio camino entre Madrid y Barcelona

Los elementos como Juan o Zaragoza no aportan ninguna información si no se integran en elementos más complejos, o sea no son proposiciones. Por lo tanto no aparecerán simbolizadas en el cálculo de Proposiciones, sí, en cambio, se representarán en el cálculo de Predicados.

En los enunciados anteriores no se pueden dividir los enunciados en elementos del lenguaje con significado propio de información. Así, si decimos "es informático" no aporta ninguna información si se prescinde del sujeto.

En cambio, hay otras frases que sí se pueden dividir en distintos elementos del lenguaje:

Llueve y me mojo

se puede descomponer en:

Llueve
Me mojo

separadas por la conectiva y.

Cada una de ellas tiene un contenido de información propio y la palabra "y" es un elemento del lenguaje que nos permite construir una nueva frase más compleja. Los enunciados con los que vamos a trabajar pueden ser atómicos, que ya los hemos visto antes y que son aquellos que no llevan partículas conectivas en su composición. También podemos encontrarnos con enunciados que sí lleven conectivas en su composición a estos enunciados se les llama moleculares.

Proposición molecular : aquella que está formada por proposiciones atómicas enlazadas por conectivas y cuyo valor de verdad resultante depende de los valores de verdad de cada una de las proposiciones atómicas y del comportamiento de los conectores que las enlazan.

La forma en que vamos a representar los enunciados, tanto atómicos como moleculares, será la siguiente:

- elegiremos una variable proposicional para cada enunciado atómico[2].
- cada proposición molecular se dividirá en tantas proposiciones atómicas como se requiera, haciendo la separación cada vez que nos encontremos con una conectiva y dividiendo la proposición en bloques.

Ejemplos:

Proposiciones atómicas:

Hoy hace sol	p

(donde *p* es una variable que representa al enunciado «Hoy hace sol»)

Mañana es lunes	q
La ciencia es la estética de la inteligencia	r
Marta es amiga de Juan	s

(Esta última es más conveniente representarla en Cálculo de Predicados que en Cálculo de Proposiciones ya que lleva un predicado relacional)

Proposiciones moleculares:

Hoy vamos al cine y mañana de paseo	$p \wedge q$
descomposición:	
Hoy vamos al cine	p
y	conectiva (conjunción '$\wedge$')
mañana vamos de paseo	q
Vamos de compras o de paseo	$p \vee q$
descomposición:	
Vamos de compras	p
o	conectiva (disyunción ' $\vee$ ')
vamos de paseo	q

No son proposiciones:

Nieve la está hoy	(no tiene sentido)
¿Lloverá mañana?	(no es posible pronunciarse sobre su significado)
Esta frase es falsa	(tampoco es posible decir si es cierta o falsa)

1.1. CONECTIVAS EN EL CÁLCULO PROPOSICIONAL

De acuerdo con el planteamiento matemático consideraremos los siguientes símbolos para representar las conectivas lógicas[3]:

- Negación ¬

[2] También podemos utilizar una palabra o un mnemónico que nos de una idea de lo que representa. Así, la sentencia "Está lloviendo" la podemos representar con la palabra *llueve* o simplemente con la variable proposicional *p*.

[3] En la bibliografía sobre lógica se utilizan distintos símbolos. Se han elegido estos por que creemos que son los más intuitivos y son utilizados por muchos autores.

- Conjunción $\wedge$
- Disyunción $\vee$
- Condicional $\rightarrow$
- Bicondicional $\leftrightarrow$

Antes de comenzar a describir cada una de ellas en particular, indicar que la negación se considera una conectiva *monádica*, ya que se aplica a una sola proposición, tanto atómica como molecular; sin embargo las demás conectivas aquí tratadas son *diádicas* o *binarias*, ya que necesitan dos enunciados para poder aplicarse.

- **NEGACIÓN: ¬ p**

Representa los elementos del lenguaje que permiten construir una frase (¬p) a partir de otra p, del tipo:

no p
es falso que p
no es cierto p, ...

Al negar un enunciado nuestra intención es decir que ese enunciado es falso. Si un enunciado es verdadero su negación es falso; y si un enunciado es falso, su negación es verdadero. Esta conectiva es la única, de las que estudiaremos, que modifica la proposición en sí al aplicarse a un único argumento; las demás lo que hacen son combinaciones de dos.

Tabla:

p	**¬ p**
F	V
V	F

Ejemplos:

Hace sol	p	
No hace sol	¬ p	
El saber no ocupa lugar	¬ q	(siendo *q* "El saber ocupa lugar")

- **CONJUNTOR: p ∧ q**

Representa la clase de elementos del lenguaje con dos proposiciones (atómicas o moleculares) que tienen la siguiente estructura:

p y q
p pero q
p sin embargo q
p no obstante q
p a pesar de q, ...

Una conjunción afirma la verdad de sus dos componentes. Es verdadera si sus dos componentes son verdaderas, y es falsa si al menos una de sus componentes lo es.

Tabla:

p	q	p ∧ q
V	V	V
V	F	F
F	V	F
F	F	F

Ejemplo:

Llueve y hace frío $p \wedge q$
(donde *p* representa la sentencia "Llueve" y *q* "Hace frío")

Un cuadrado tiene los cuatro lados iguales y los ángulos rectos $r \wedge s$
(donde *r* representa el enunciado "Un cuadrado tiene los cuatro lados iguales" y *s* "Un cuadrado tiene los cuatro ángulos rectos")

- **DISYUNTOR: p ∨ q**

Representa la clase de elementos del lenguaje que a partir de dos proposiciones construyen una nueva con la estructura:

o p o q o ambas cosas
al menos p o q
como mínimo p o q, ...

El significado del disyuntor es: la disyunción de dos proposiciones es verdadera cuando una al menos de esas dos proposiciones es verdadera y por supuesto cuando ambas lo son; es falsa, en cambio, sólo cuando ambas son falsas.

Tabla:

p	q	p ∨ q
V	V	V
V	F	V
F	V	V
F	F	F

El significado del disyuntor sólo coincide parcialmente con el «o» del lenguaje ordinario. En lenguaje ordinario la partícula «o» tiene dos sentidos:

a) Sentido exclusivo, según el cual la disyunción establece que uno de sus miembros es verdadero y el otro falso, con lo que se excluye, por tanto, la posibilidad de una simultánea verdad de ambos.

Ejemplo:

Ana nació en Elche o Alcoy $p \vee q$
(se excluye la posibilidad de que los dos sean verdaderos)

b) Sentido inclusivo, se da la posibilidad de que las dos opciones sean verdaderas, es decir, se indica que al menos una de esas proposiciones sea verdadera y no se dice nada acerca de la otra, pudiendo ser también verdadera.

Ejemplo:

Para estudiar informática es necesario saber Inglés o Francés $p \vee q$
(pueden saber uno de los dos idiomas pero no se excluye la posibilidad de que sepan ambos idiomas)

Los polígonos irregulares tienen los lados o los ángulos desiguales $r \vee s$

En lógica de juntores o conectores la disyunción que vamos a considerar a partir de este momento es la «o» inclusiva. Luego el sentido de $p \vee q$ es:

"Para que sea verdadera la disyunción, tiene que ser al menos una de las dos proposiciones verdadera, pero también lo pueden ser ambas"

- **IMPLICADOR: $p \rightarrow q$**

La expresión que precede al implicador se llama **antecedente** y la que le sucede, **consecuente**. Es adecuado para representar la relación de causa/efecto que puede construirse en el lenguaje usual de muchas formas distintas:

si p entonces q
p sólo si q
q si p
q necesario para p
p suficiente para q
no p a menos que q, ...

Ejemplo:

Si el perro ladra, entonces alguien ha entrado a casa
Ladra el perro sólo si alguien entra a casa
Alguien ha entrado a casa si ladra el perro
Que alguien entre a casa es necesario para que el perro ladre
Que el perro ladre es suficiente para que alguien haya entrado a casa
No ladra el perro a menos que alguien entre a casa

El sentido del implicador es: Una expresión es verdadera siempre que no se dé el caso de que el antecedente sea verdadero y el consecuente falso; y falsa cuando se da ese caso. Conlleva una condición suficiente de izquierda a derecha. Para que una cosa

sea condición suficiente de otra, lo único que hace falta es que, de ser verdadera o real la 1ª, lo sea también la 2ª. Parece que decimos dos cosas distintas pero es lo mismo, pues al decir "si p entonces q" parece que se va de izquierda a derecha y al decir "p sólo si q" parece que se va de derecha a izquierda; pero en ambos casos se va de izquierda a derecha. Se puede ver mejor en algunas frases si añadimos "es que".

Tabla:

p	**q**	**p → q**
V	V	V
V	F	F
F	V	V
F	F	V

Ejemplos:

Si Jassan II se preocupa por sus súbditos, yo soy fraile $p \rightarrow q$

Formalización:

Jassan II se preocupa por sus súbditos p

yo soy fraile q

Si Luis es peripatético, cree en la materia y en las formas $p \rightarrow q$

Se puede poner:

Luis es peripatético si **es que** cree en la materia y en las formas

Si una persona es marxista, admite la verdad del materialismo histórico $p \rightarrow q$

Una persona es marxista **sólo si (es que)** admite la verdad del materialismo histórico

A quien Dios se la dé, San Pedro se la bendiga $p \rightarrow q$

En martes, ni te cases ni te embarques $p \rightarrow \neg q \wedge \neg r$

Sean los dos enunciados que tenemos a continuación:

Los perros son vertebrados p

La Tierra es un planeta q

Sabemos que p y q son enunciados verdaderos, por tanto si realizamos las siguientes implicaciones:

1.- $p \rightarrow q$

2.- $p \rightarrow \neg q$

3.- $\neg p \rightarrow q$

4.- $\neg p \rightarrow \neg q$

Sólo la 2ª ($p \rightarrow \neg q$) es falsa, porque en ella se da que el antecedente es verdadero y el consecuente es falso. Las demás son verdaderas, ya que no se tiene en cuenta el contenido de la proposición si no sólo su valor de verdad. Por tanto tendría valor verdadero la siguiente proposición: «Si los perros son vertebrados, entonces la Tierra es un planeta»

No hay que confundir la implicación *si ... entonces* con la idea de que el contenido del antecedente tenga relación con el contenido del consecuente. Esto es porque en Lógica de Proposiciones no se tiene en cuenta el contenido de las mismas, sino tan sólo su valor de verdad.

Comentario: en la antigüedad se sostenía una interesante discusión; Filón de Megara decía que para que dos proposiciones se impliquen basta con que no se dé el caso de que la 1ª sea verdadera y la 2ª falsa, sin importar, como ya se ha dicho, el contenido de las mismas. Se llama **implicación material**. Por ejemplo, veamos la siguiente proposición, en la que los antiguos filósofos no estaban muy de acuerdo:

«Si es de noche, entonces discuto»

esta proposición es verdadera cuando:
- sea de día aunque no discuta (implicación con antecedente falso)
- discuto aunque no sea de noche (implicación con consecuente V)

Diodoro Crono, maestro de Filón, no aceptaba este punto de vista, porque le parecía absurdo que la proposición condicional "si es de noche, entonces discuto" se convirtiese circunstancialmente V durante el día. Para Diodoro si una proposición es V lo tiene que ser siempre, esto es, es imposible que se de el caso de que el antecedente sea V y el consecuente F (implicación material en todo momento o estricta). La implicación diodórica es lo que hoy se define como **implicación formal**.

- **COIMPLICADOR: $p \leftrightarrow q$**

Representa la clase de frases del lenguaje que se interpretan por alguno de los siguientes esquemas:

p si y sólo si q
p cuando y solamente cuando q
p equivale a q, ...

Al coimplicador también se le llama *equivalencia* o *bicondicional*. Una coimplicación es verdadera cuando sus dos componentes tienen el mismo valor de verdad, esto es, cuando ambos son verdaderos o ambos falsos; y es falsa en caso contrario.

Tabla:

p	q	$p \leftrightarrow q$
V	V	V
V	F	F
F	V	F
F	F	V

Ejemplo:

Dos números son opuestos si y sólo si tienen el mismo valor absoluto y distinto signo $p \leftrightarrow q \wedge r$

El nº 2 es el menor de todos los pares p
El nº 2 es el menor de todos los nº primos q

1.- $p \leftrightarrow q$
2.- $p \leftrightarrow \neg q$

3.- $\neg p \leftrightarrow q$
4.- $\neg p \leftrightarrow \neg q$
La 1ª y la 4ª son verdaderas, y la 2ª y 3ª son falsas.

1.2. FÓRMULAS PROPOSICIONALES

Fórmula Proposicional : Toda sentencia ordenada, formada por variables proposicionales, conectivas y eventualmente paréntesis.

Fórmula Proposicional Bien Formada (fbf) : Decimos que una fórmula proposicional es una fbf si se obtiene a partir de las reglas siguientes:

1) Una variable proposicional es una fbf.

2) Si p es una fbf entonces $\neg p$ también lo es.

3) Si p y q son fbf también lo son $p \wedge q$, $p \vee q$, $p \rightarrow q$, $p \leftrightarrow q$.

4) Diremos que una fbf es aquella que se obtiene por aplicación finita de los pasos 1, 2 y 3.

Ejemplos:

fbf	no fbf
$p \wedge q \rightarrow r$	$p \neg r$
$\neg p \vee q$	$p \rightarrow \wedge r$
$s \leftrightarrow \neg(r \wedge s)$	$\neg(r \vee \rightarrow q)$
$\neg p \rightarrow r \wedge s$	$p \neg (r \wedge s)$

El **Lenguaje Formal de Primer Orden para el Cálculo de Proposiciones** es el de las *fórmulas proposicionales bien formadas*.

En el lenguaje usual las proposiciones afectadas por las conectivas se sobreentienden por el contenido de las frases o se explicitan por la utilización de símbolos de puntuación (coma, punto y coma, ...). En el lenguaje formalizado, de igual manera, es necesario el uso de paréntesis para acotar las proposiciones moleculares afectadas por cada conectiva. Para ser rigurosos deberíamos incluir un punto en la definición anterior de *fbf*:

1') Si p es una fbf entonces (p) también lo es.

Para evitar el exceso de paréntesis se define una jerarquía entre las conectivas:

- nivel 1: $\neg$
- nivel 2: $\wedge$, $\vee$
- nivel 3: $\rightarrow$, $\leftrightarrow$

Vamos a dar algunas reglas prácticas para eliminar el exceso de paréntesis, evitar la ambigüedad de interpretación de las fórmulas así como para determinar la asociatividad entre conectivas del mismo nivel. Adoptaremos los siguientes convenios. Si A y B son dos fbf se establece que:

1.- La expresión A & B representa la fbf (A & B), siendo & cualquier conectiva binaria. (Esto vendría dado por el punto 1').

2.- La expresión $A \wedge B \rightarrow C$ representa a la $(A \wedge B) \rightarrow C$. Análogamente para $A \vee B \rightarrow C$, $C \rightarrow A \wedge B$, $C \rightarrow A \vee B$. (Esto ya viene dado al definir la jerarquía entre prioridades).

3.- Igual que en 2 sustituyendo $\rightarrow$ por $\leftrightarrow$.

4.- La expresión $A \rightarrow B \rightarrow C$ representa a la $A \rightarrow (B \rightarrow C)$, aunque es aconsejable utilizar siempre esta última. (Esto viene dado por la propiedad de asociatividad por la derecha de la conectiva $\rightarrow$)

5.- La expresión $A \wedge B \vee C$ representa a la $(A \wedge B) \vee C$, aunque es aconsejable utilizar siempre esta última. Lo mismo si intercambiamos las conectivas $\wedge$ y $\vee$.

Ejemplos:

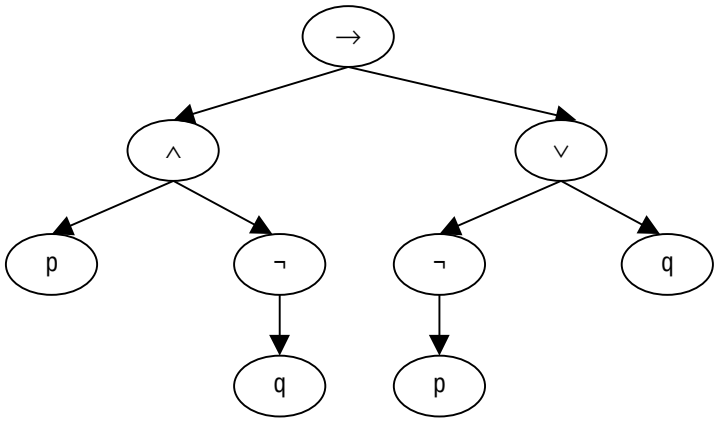
$(\neg p) \vee (\neg q)$	se representa por	$\neg p \vee \neg q$
$((p \wedge \neg q) \rightarrow (\neg p \vee q))$	se representa por	$p \wedge \neg q \rightarrow \neg p \vee q$
$((\neg(p \leftrightarrow q)) \leftrightarrow ((p \wedge \neg q) \vee q))$	se representa por	$\neg(p \leftrightarrow q) \leftrightarrow (p \wedge \neg q) \vee q$

De todas formas, un consejo práctico a la hora de escribir fórmulas es que, ante la duda, es aconsejable utilizar paréntesis, aunque no sea estrictamente necesario, para así reducir la posibilidad de error.

Ejemplo:

Para ver claramente la estructura sintáctica de una fórmula nos puede ayudar dibujar la fbf en forma de árbol etiquetado. Así el árbol sintáctico de la fórmula bien formada anterior sería:

$p \wedge \neg q \rightarrow \neg p \vee q$



Podemos comprobar fácilmente que las hojas del árbol serán siempre variables proposicionales y los nodos internos corresponderán a las conectivas lógicas.

2. EL LENGUAJE DEL CÁLCULO DE PREDICADOS

El lenguaje formal para el Cálculo de Predicados es una generalización del visto en Cálculo de Proposiciones. En Lógica de Proposiciones se analizaban formalmente unidades de información cuya estructura se contemplaba como un todo, sin diferenciar sus componentes. Este planteamiento no permite representar matemáticamente determinadas estructuras deductivas, que son correctas en el lenguaje usual.

Ejemplo:

Representación en el Cálculo de Proposiciones visto hasta ahora:	
- Sólo los portugueses se dejan engañar por los vendedores ambulantes	$v \rightarrow p$
- Antonio se deja engañar por María	q
- Antonio no es portugués.	¬r
Luego, María no es una vendedora ambulante.	¬s

Según esta representación ninguna de las proposiciones (p, v, r, q y s) describen lo que intuitivamente se ve que tienen en común unas con otras, luego la relación entre premisas y conclusión, no puede establecerse en este nivel de representación.

Esto se debe a que ahora la relación entre las proposiciones está en las propiedades que aparecen en ellas: se afirman en ellas las mismas propiedades o relaciones para distintas personas o conjunto de personas. Hay que definir ahora un

cálculo que no tome como base la simbolización matemática de la proposición total sino la de sus componentes:

- *Qué se afirma* ⇨ **predicado**
- *De quién o quienes se afirma* ⇨ **sujeto** o **término**

Ejemplos:

María es rubia	sujeto:	María
	predicado:	ser rubia
Juan se sienta entre Pedro y Antonio	predicado:	sentarse entre
	sujetos:	Juan, Pedro y Antonio

Lógica de Predicados : parte de la Lógica que destaca el contenido de las sentencias que forman parte de los argumentos, así como la forma de los argumentos.

Por otro lado, no dejamos la Lógica de Proposiciones, sino que la ampliamos, y trabajamos con sus resultados. Los elementos fundamentales del cálculo de predicados son los símbolos de **variables**, de **constantes**, de **funciones** y de **predicados**.

Término : será o bien un símbolo de constante, o de variable o valor devuelto por una función; y para simbolizarlos se supone un dominio genérico (*Universo del discurso*), no vacío en el cual van a tomar valores:

- los símbolos de *constante* representan objetos o entes concretos del dominio. Estos objetos pueden ser físicos, personas, conceptos, etc.

- los símbolos de *variable* son términos que nos permiten ser indefinidos respecto al ente al que se refieren. Representan a cualquier elemento del dominio.

- los símbolos de *función* denotan una función que aplicada a cualquier elemento del dominio de discurso da como resultado otro elemento del domino.

Predicado se usa para representar una propiedad o relación entre elementos del dominio.

Tipos de predicados:

1.- *Absolutos* o *monádicos*: se refieren a un sólo término. Normalmente hacen referencia a propiedades o características de los objetos.

2.- *De Relación* o *Poliádicos*: se refieren a varios sujetos (*diádicos, triádicos,...*). Suelen representar relaciones entre objetos.

Generalizando, podemos considerar a las constantes como funciones sin argumentos. De la misma forma podemos decir que una proposición es un predicado sin argumentos.

Si queremos aplicar la Lógica para decir cosas sobre un mundo imaginable o real, los símbolos de términos y los de predicados deben estar relacionados con cosas de dicho mundo:

objetos del mundo	⇨	términos en lógica
relaciones del mundo	⇨	predicados en lógica

Representación:

Predicados: Se utilizarán o bien palabras mnemónicas (escribió, casados, ...) o bien las letras mayúsculas (P, Q, R, ...) que pueden ir acompañadas de cifras o subíndices (P1, Q3, ..., P_1, P_2).

Constantes: Se utilizarán palabras mnemónicas (jorge, andrés, ...) o bien las letras iniciales del abecedario (a, b, c, ...) que pueden ir acompañadas de cifras o subíndices (a1, b_2, ...). No puede haber confusión entre una constante y el nombre de un predicado pues es el propio contexto el que las diferencia.

Variables: Se utilizarán las letras finales del abecedario (x, y, z, ...). Pueden llevar subíndices (x_1, x_2, ...).

Funciones: Se utilizarán las letras f, g, h, ... con o sin subíndices.

Ejemplo:

Representar en Cálculo de Predicados la sentencia «Voltaire escribió Cándido». Podemos escribir:

escribió(voltaire, cándido)

o bien

P(a, b)	donde	P(x,y)	x escribió la obra y
		a	Voltaire
		b	Cándido

Este tipo de fórmulas son *atómicas o elementales*, porque no llevan en su estructura ninguna conectiva. Las fórmulas atómicas pueden combinarse usando las conectivas dando lugar a fórmulas más complejas (*fórmulas moleculares)*, tal como ya se vio en cálculo de proposiciones. Las conectivas lógicas tienen el mismo significado.

Ejemplo:

«Si el coche es de Juan, entonces el coche es verde»
pertenece (coche, juan) $\rightarrow$ color (coche, verde)

En general para representar una sentencia, en cálculo de predicados, nos fijamos en los objetos que intervienen y la relación que describe la sentencia y los representaremos mediante predicados y términos. Frecuentemente el predicado se identifica con el verbo de la sentencia y los términos con el sujeto. Normalmente tendremos distintas opciones para representar una misma sentencia. El que elige la representación decide el vocabulario para los predicados y términos que va a utilizar y define lo que significan los elementos de ese vocabulario. La simplicidad de la representación dependerá del uso que se quiera hacer del conocimiento contenido en dichas frases. Para tomar una decisión de diseño lo más acertada posible nos puede ayudar el conocer todas las sentencias con las que trabajaremos para poder entresacar sus relaciones.

Ejemplo:

«La casa es amarilla»

Representación: 1.- amarillo(casa).
2.- color (casa, amarillo)
3.- valor(color, casa, amarillo)

Otro aspecto nuevo es el uso de funciones. Una función es un tipo de relación y por tanto podría representarse por medio de predicados. La diferencia estriba en que su aplicación, en lugar de dar como resultado un valor de verdad, nos proporciona un objeto del universo de discurso. De ahí que una función de *n* argumentos pueda ser reemplazada por una predicado de *n*+1 argumentos, de forma que no perdemos expresividad si prescindimos de ellas.

Ejemplo:

«Félix es el padre de Clara»

1.- Representación mediante funciones: padre(clara) equivale a felix

2.- Representación sin funciones: padreDe(felix,clara)

2.1. CUANTIFICACIÓN

Su nombre procede de que son usados para expresar la cantidad de objetos que satisfacen alguna condición. Las sentencias cuantificadas nos indican cuantos individuos verifican una determinada propiedad o entre cuantos individuos se da una determinada

relación. Hay dos tipos de sentencias cuantificadas: *universales y existenciales.* Veamos cada una de ellas:

- **Cuantificador Universal (∀)** indica que todos los elementos del universo de discurso cumplen una determinada propiedad. Representa a las sentencias del tipo:
  - Para todo x, x es tal que ...
  - Para cada x, x es tal que ...

- **Cuantificador existencial (∃)** indica que existe por lo menos un individuo del universo que verifica una determinada propiedad o relación. Representa las sentencias con el siguiente esquema:
  - Existe un x, tal que ...
  - Hay un x, tal que ...
  - Para algún x, x es tal que ...
  - Hay por lo menos un x tal que ...

Veamos algunas definiciones relacionadas con la cuantificación:

Índice de un cuantificador : variable adosada al cuantificador.

Prefijo cuantificacional : cuantificador e índice cuantificacional.

Matriz cuantificacional : parte de la fórmula afectada por el prefijo cuantificacional.

Ejemplo:

$$\forall x \; P(x)$$

Prefijo:	$\forall x$
Índice:	x
Matriz:	P(x)

Un prefijo puede tener varios cuantificadores con sus índices correspondientes y una matriz puede encerrar cuantificadores a su vez.

Ejemplo:

$$\forall x \, \exists z \; [P(x) \vee Q(x) \vee \exists y \, Q(y) \vee P(z)]$$

Alcance o **Ámbito** cuantificacional : hace referencia a la parte de la fórmula a la que alcanza el cuantificador, es decir, sobre la que ejerce su cuantificación.

Ejemplo:

$$\forall x \, \exists y \, \forall z \, \forall s \; P(x, y, z, s)$$

se entiende como:

$$\forall x \, \{ \exists y \, [\forall z \, (\forall s \; P(x, y, z, s))] \}$$

Variable libre : variable que aparece en la fórmula y no está afectada por ningún cuantificador.

Variable ligada : variable afectada, y por tanto asociada, por algún cuantificador.

Ejemplo:

$\forall x \quad P(x, y)$

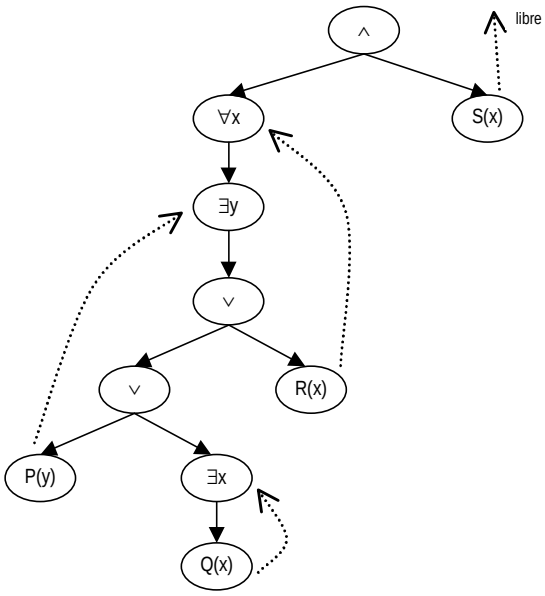
variable ligada: x
variable libre: y

Si una variable se encuentra dentro del alcance de dos cuantificadores que la llevan adosada como índice, esta variable queda ligada al cuantificador más cercano al predicado donde aparece, es decir el de menor alcance. Así podemos enunciar la siguiente regla de proximidad de cuantificadores: cuando trabajemos con fórmulas que varios cuantificadores el proceso de cuantificación se realiza en el orden de *mayor a menor proximidad* a la fórmula cuantificada.

Ejemplo:

$\forall x \, \exists y \;\; [\, P(y) \vee \exists x \, Q(x) \vee R(x) \,] \wedge S(x)$

La variable *x* del predicado *Q* está dentro del alcance de dos cuantificadores, $\forall x$ y $\exists x$, pero queda ligada al $\exists$ por estar más cercano. En cambio, la variable *x* del predicado *R* sólo está dentro del ámbito del $\forall x$, y por tanto está ligado a este cuantificador. Por otro lado, la variable *x* del predicado *S* está fuera del ámbito tanto del $\forall x$ como del $\exists x$, por lo que es una variable libre. Esto se ve más fácilmente si obtenemos el árbol sintáctico:



El cambio de orden cuantificacional puede alterar el significado de la frase. Así, aunque se puede cambiar el orden cuando los cuantificadores son del mismo tipo (todos universales o todos existenciales) no ocurre lo mismo cuando hay universales junto con existenciales.

Ejemplo:

Para cualquier número natural *x* existe al menos otro *y* tal que $x < y$ (cierto)

$\forall x \exists y \; M(x, y)$

Existe al menos un número natural *y* tal que para cualquier *x*, $x < y$ (falso)

$\exists y \forall x \; M(x, y)$

Dominio o **Universo del Discurso** : conjunto de objetos que constituye el marco de referencia de nuestro lenguaje en un momento dado, o dicho de otro modo, dominio en el cual se interpretarán nuestros términos.

Así, cuando tenemos un universo del discurso finito, el cuantificador universal se puede considerar como una generalización de la conjunción y el cuantificador existencial como una generalización de la disyunción.

Ejemplo:

Sea el dominio $D = \{ a_1, a_2, ..., a_n \}$, tenemos que:

$\forall x \, P(x)$	es equivalente a	$P(a_1) \wedge P(a_2) \wedge ... \wedge P(a_n)$
$\exists x \, P(x)$	es equivalente a	$P(a_1) \vee P(a_2) \vee ... \vee P(a_n)$

La elección del Universo del Discurso al trabajar con el Cálculo de Predicados es importante, hasta el punto de condicionar:

- la formalización del predicado:

«Todas las estudiantes saben Informática»

dominio: las estudiantes	$\forall x \; I(x)$
dominio : las personas	$\forall x \; (E(x) \rightarrow I(x))$

- el valor de verdad del predicado:

P(x) : "x es mayor que 0"

dominio: los nº enteros	$\forall x \; P(x)$	↬	falso
dominio: los nº naturales	$\forall x \; P(x)$	↬	verdadero

Por tanto, antes de formalizar una sentencia en el Cálculo de Predicados debemos definir nuestro Universo del Discurso.

2.2. FÓRMULAS DEL CÁLCULO DE PREDICADOS

Finalmente vamos a dar las definiciones que nos permitirán dejar claro cuál es el lenguaje que utilizaremos para el Cálculo de Predicados. La mayoría son una simple extensión de las dadas para el Cálculo de Proposiciones.

Alfabeto : consta de los siguientes símbolos:

a) Símbolos de términos:

variables	x, y, z, ...
constantes	a, b, c, ...
funciones	f, g, h, ...

b) Símbolos de predicado P, Q, R, ...

c) Símbolos de conectivas y puntuación: $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, (,)$

d) Símbolos de cuantificación:

cuantificador universal	$\forall$
cuantificador existencial	$\exists$

Fórmula Bien Formada (fbf) : es una sucesión de símbolos del alfabeto que verifica las reglas de formación siguientes:

1.- Toda proposición (variable proposicional) es una fórmula bien formada.

2.- Si P es un predicado, entonces $P(t_1, t_2, ..., t_n)$ es una fbf, siendo t_i términos.

3.- Si F es una fórmula bien formada que tiene la variable x_i libre, entonces:

$\forall x_i \; F [x_1, x_2, ..., x_i, ..., x_n]$

$\exists x_i \; F [x_1, x_2, ..., x_i, ..., x_n]$

son fbf. La variable x_i queda ligada al cuantificador introducido y las otras variables x_k de F distintas de x_i siguen como antes.

4.- Si A y B son fbf entonces $\neg A$, $A \wedge B$, $A \vee B$, $A \rightarrow B$ y $A \leftrightarrow B$ son fbf.

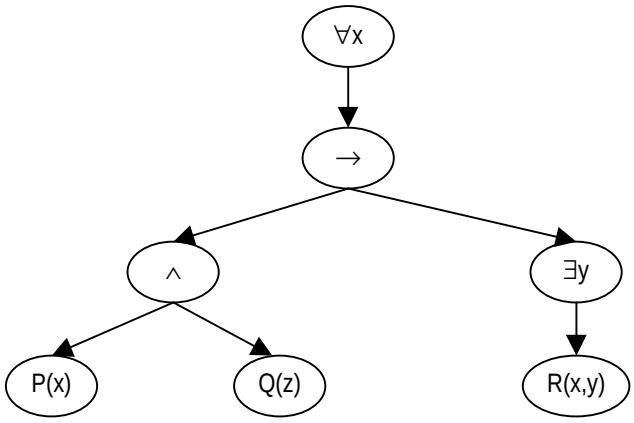
5.- Sólo son fórmulas bien formadas las construidas desde 1 hasta 4.

Ejemplo: La siguiente fórmula es sintácticamente correcta

$$\forall x \ (\ P(x) \wedge Q(z) \ \rightarrow \ \exists y \ R(x, y) \ )$$

ya que son fbf	R(x, y)	por 2
	$\exists y$ R(x, y)	por 3
	P(x)	por 2
	Q(z)	por 2
	$P(x) \wedge Q(z)$	por 4
	$P(x) \wedge Q(z) \rightarrow \exists y$ R(x, y)	por 4
	$\forall x$ ($P(x) \wedge Q(z) \rightarrow \exists y$ R(x, y))	por 3

Una forma gráfica de ver la estructura de una fórmula es mediante el árbol sintáctico:



Fórmula atómica o **átomo** : consiste en un predicado con todos sus argumentos instanciados por términos (bien por variables, constantes o funciones).

Literal : es un átomo (*literal positivo*) o la negación de un átomo (*literal negativo*).

Cláusula : es una disyunción de literales.

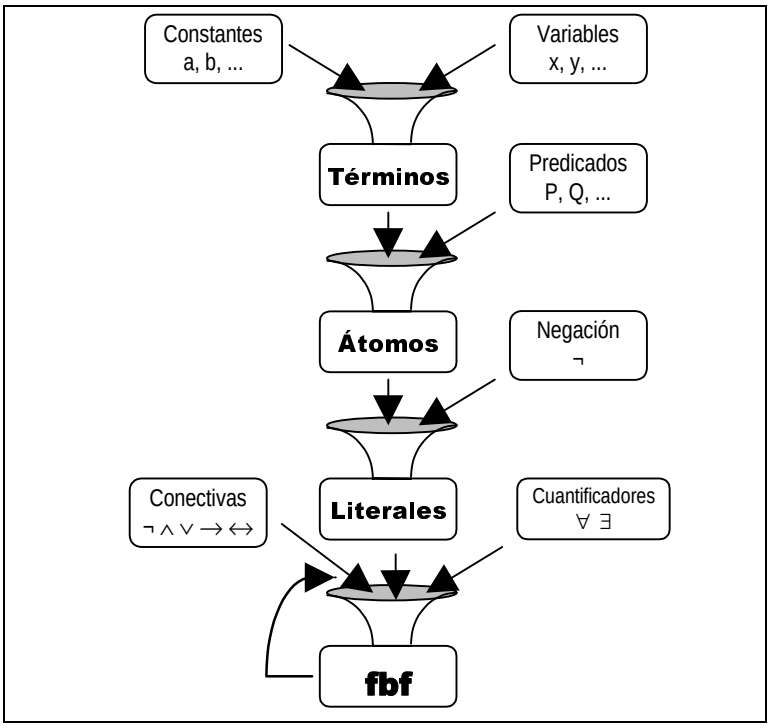


Ilustración 1: Vocabulario de la lógica de primer orden

Ejemplos:

«Sólo los informáticos entienden a los informáticos»
$\forall x \forall y [I(x) \wedge E(y, x) \rightarrow I(y)]$

«Nadie entraba en la academia de Platón a menos que supiera Geometría»
$\forall x [A(x) \rightarrow G(x)]$

En el lenguaje que nosotros vamos a trabajar los cuantificadores se aplicaran exclusivamente a las variables de términos. Es lo que denominamos

Cálculo de Predicados de Primer Orden.

Cuando aparecen fórmulas con los cuantificadores aplicados a símbolos de predicados y funciones trabajamos en **Cálculo de Predicados de Segundo Orden**. Esto supone la definición de Dominios de Predicados y Funciones asociados a estas variables. Dado que en la actualidad no se conocen procedimientos que permitan decidir la validez o no de las estructuras deductivas en cálculos de orden superior, el libro se limita a trabajar con el cálculo de primer orden.

Cálculo de Proposiciones	⇎	*no se permiten variables de ningún tipo*
Cálculo de Predicados	⇎	*las variables sólo pueden representar objetos*
Cálculo de Segundo Orden	⇎	*las variables pueden representar predicados*

2.3. ENUNCIADOS CUANTIFICADOS

Los enunciados cuantificados con los que vamos a trabajar van de ser de diferentes tipos:

- *Universales afirmativos*: todos los *x* verifican *P* $\forall x\ P(x)$
- *Universales negativos*: ningún *x* es *P* $\forall x\ \neg P(x)$
- *Existenciales afirmativos*: algún *x* es *P* $\exists x\ P(x)$
- *Existenciales negativos*: algún *x* no es *P* $\exists x\ \neg P(x)$

Los cuantificadores universal y existencial se relacionan entre sí gracias a la conectiva de negación, de forma que podemos pasar de uno a otro:

$\neg\forall x\ P(x)$ (no todos los *x* tienen la propiedad *P*) ⇔
⇔ (hay algún *x* que no tiene la propiedad *P*) $\exists x\ \neg P(x)$

$\forall x\ \neg P(x)$ (todos los *x* poseen la propiedad no *P*) ⇔
⇔ todos los *x* carecen de la propiedad *P* ⇔
⇔ (no existe ningún *x* que posea la propiedad *P*) $\neg\exists x\ P(x)$

Estas equivalencias anteriores las podemos ver como generalizaciones de las leyes de De Morgan y representan la dualidad entre los cuantificadores. La regla para las sentencias que tienen un símbolo de negación es que si uno de los cuantificadores aparece con un símbolo negador a su izquierda, puede ser sustituido por el otro cuantificador con la negación a su derecha (o al revés), quedando una expresión resultante equivalente a la inicial. De igual forma, cuando tengamos una expresión con cuantificador y dos símbolos de negación:

$\neg\exists x\ \neg P(x)$ (no hay ningún *x* que posea la propiedad no *P*) ⇔
⇔ no hay ningún *x* que no sea *P* ⇔
⇔ (todos los *x* verifican *P*) $\forall x\ P(x)$

$\neg\forall x\ \neg P(x)$ (no todos los *x* carecen de la propiedad *P*) $\Leftrightarrow$
$\Leftrightarrow$ (hay algún *x* que tiene la propiedad *P*) $\exists x\ P(x)$

Habitualmente al formalizar sentencias de lenguaje natural, las expresiones cuantificada universalmente van acompañadas de la conectiva de la implicación ya que de esta manera restringimos el dominio de la variable cuantificada universal. Así, por ejemplo:

$$\forall x\ [\ P(x) \rightarrow Q(x)\ ]$$

significa que todos los *x*, si verifican la propiedad *P* entonces verifican la propiedad *Q*, es decir, no todos necesariamente cumplen la propiedad *Q*, pero sí al menos aquellos que cumplen la propiedad *P*. Complementariamente, una expresión cuantificada existencialmente suele ir seguida, en general, de la conectiva de la conjunción. De esta forma:

$$\exists x\ [\ P(x) \wedge Q(x)\ ]$$

significa que existe algún *x* que verifica *P* y que verifica también *Q*. Las relaciones que existen entre los esquemas anteriores son, al igual que en las expresiones sin conectivas, a través del negador de la siguiente forma:

$\neg\forall x\ [\ P(x) \rightarrow Q(x)\ ]$ $\Leftrightarrow$
$\Leftrightarrow$ $\exists x\ \neg[\ P(x) \rightarrow Q(x)\ ]$ $\Leftrightarrow$
$\Leftrightarrow$ $\exists x\ \neg[\ \neg P(x) \vee Q(x)\ ]$ $\Leftrightarrow$
$\Leftrightarrow$ $\exists x\ [\ P(x) \wedge \neg Q(x)\ ]$

$\neg\exists x\ [\ P(x) \wedge Q(x)\ ]$ $\Leftrightarrow$
$\Leftrightarrow$ $\forall x\ \neg[\ P(x) \wedge Q(x)\ ]$ $\Leftrightarrow$
$\Leftrightarrow$ $\forall x\ [\ P(x) \rightarrow \neg Q(x)\ ]$

3. LIMITACIONES EXPRESIVAS DE LA LÓGICA DE PRIMER ORDEN

Hasta aquí hemos visto cómo traducir sentencias del lenguaje natural, en nuestro caso castellano, al lenguaje de la Lógica de Primer Orden (lo que hemos llamado *fbf* - "fórmulas bien formadas"). Por tanto, podemos analizar en este momento

las limitaciones de dicho lenguaje, y veremos como muchas expresiones del lenguaje no pueden ser trasladadas a fórmulas lógicas.

Una de las limitaciones del lenguaje visto para la lógica aparece al intentar trabajar con sentencias numéricas, ya que nos encontramos con un amplio abanico de posibilidades de cuantificación en el lenguaje natural, frente a dos únicos cuantificadores en lógica. La incorporación a nuestro lenguaje lógico del símbolo de igualdad nos permitirá obtener una mayor potencia expresiva para el tratamiento de sentencias numéricas, pero sin llegar a alcanzar la riqueza expresiva del lenguaje natural o del lenguaje matemático.

Ejemplos:

Unicamente un alumno de clase ha nacido en Alcoy

Tenemos dos cubos grandes

La mayoría de los alumnos de la clase saben informática

Otra clara limitación viene dada por el uso de los tiempos, ya que la Lógica de Primer Orden no permite el cambio de las relaciones en el tiempo, mientras que en lenguaje natural podemos utilizar el pasado, el presente o el futuro.

Ejemplo:

Hoy hace calor pero ayer hacía frío

También la Lógica de Primer Orden tiene claras limitaciones a la hora de representar la riqueza modal del lenguaje natural, así como para simbolizar incertidumbre. Por estas y otras razones, han ido apareciendo en los últimos tiempos distintas extensiones o desviaciones de la lógica que pretenden paliar, en lo posible, estas deficiencias: lógicas modales, lógicas temporales, lógicas no monótonas, lógicas polivalentes, lógica difusa, ... Pero dichas lógicas escapan a las pretensiones de este libro.

4. EJERCICIOS

- Representar en *lenguaje proposicional* los siguientes enunciados:

1.- **Si acepto el mundo que me ofrecen y soy feliz así, entonces empiezo a cavar mi propia sepultura; o bien, si no soy feliz así, y no veo tampoco posibilidad de cambiar este mundo, emprendo así mismo mi propio enterramiento.**

Solución:

Acepto el mundo que me ofrecen	p
Soy feliz así	q
Empiezo a cavar mi sepultura	r
No soy feliz así	¬q
No veo posibilidad de cambiar el mundo	¬s
Emprendo mi enterramiento	r

FBF: $[(p \wedge q) \rightarrow r] \vee [(\neg q \wedge \neg s) \rightarrow r]$

2.- **Es agradable caminar bajo la lluvia, siempre que se tenga algo suficientemente triste en que pensar.**

Solución:

Podemos poner esta otra forma que es equivalente a ella:
«Si se tiene algo suficientemente triste en que pensar entonces es agradable caminar bajo la lluvia»

Tener algo suficientemente triste en que pensar	p
Es agradable caminar bajo la lluvia	q

FBF: $p \rightarrow q$

3.- a) **Si acierto una quiniela, me haré rico.**

Solución:

Acierto una quiniela	p
Me haré rico	q

FBF: $p \rightarrow q$

b) **Sólo si acierto una quiniela me haré rico.**

Solución:
Se invierte el sentido de la frase, porque ahora acertar una quiniela es condición necesaria aunque no suficiente para hacerme rico:

FBF: $q \rightarrow p$

4.- **De haber tomado medidas en su momento, no se hubieran propagado los incendios forestales.**

Solución:

Haber tomado medidas en su momento	p
No se hubieran propagado los incendios forestales	¬q

FBF: $p \rightarrow \neg q$

5.- **Sólo si estudias aprobarás.**

Solución:

Estudias	p
Aprobarás	q

FBF: $q \rightarrow p$

6.- **En el mundo habrá paz cuando y sólo cuando no haya guerra.**

Solución:

En el mundo habrá paz	p
No haya guerra en el mundo	¬q

FBF: $p \leftrightarrow \neg q$

7.- **El que haya progreso equivale a que no haya analfabetismo.**

Solución:

Haya progreso	p
No haya analfabetismo	¬q

FBF: $p \leftrightarrow \neg q$

8.- **Si Frankestein cruza nuestras calles, ha de indicar qué y cuantos fines persigue, y si miente, le daremos con las puertas en las narices, pero si dice la verdad, le invitaremos a cenar.**

Solución:

Frankestein cruza nuestras calles	p
Indicar qué fines persigue	q
Indicar cuántos fines persigue	r
Dice la verdad (miente)	s (¬s)
Darla con la puerta en la narices	t
Le invitaremos a cenar	w

FBF: $p \rightarrow q \wedge r \wedge (\neg s \rightarrow t) \wedge (s \rightarrow w)$

9.- **Si se ganan las elecciones y nuestros representantes acceden al poder, confiaremos en ellos si y sólo si, cumplen sus promesas y el poder no les corrompe.**

Solución:

Se ganan las elecciones	p
Nuestros representantes acceden al poder	q
Confiar en ellos	r
Cumplen sus promesas	s
El poder no les corrompe	¬t

FBF: $p \wedge q \rightarrow (r \leftrightarrow s \wedge \neg t)$

10.- Formalizar en el lenguaje proposicional la sentencia del lenguaje algorítmico:

si (condición) **entonces**
acción1
sino
acción2
fsi

Solución:

condición	p
acción1	q
acción2	r

FBF: $(p \rightarrow q \wedge \neg r) \wedge (\neg p \rightarrow \neg q \wedge r)$

- Representar en *Cálculo de Predicados* las siguientes expresiones:

11.- **Antonio es médico**

Solución: 1ª versión: medico (antonio)
2ª versión: M(a) Antonio se representa por la constante *a*
Ser médico se representa por el predicado *M*

12.- **Algunos son amigos de Luis**

Solución: «Algún *x* es amigo de Luis»
Representación: l Luis (constante)
A(x, y) *x* es amigo de *y*
Fórmula: $\exists x\ A(x, l)$

13.- **Todos son hermanos**

Solución: «Todos los *x* son hermanos de todos los *y*»
Representación: H(x, y) *x* es hermano de *y*
Fórmula: $\forall x\ \forall y\ H(x, y)$

14.- a) **Algunos peces son amarillos**

Solución: «Existen individuos del dominio que son peces y amarillos»
Representación: P(x) *x* es pez
A(x) *x* es amarillo
Fórmula: $\exists x\ (P(x) \wedge A(x))$

b) **Todos los peces son amarillos**

Solución: «Para todo *x*, si *x* es pez entonces es amarillo»
Fórmula: $\forall x\ (P(x) \rightarrow A(x))$

15.- **Algunos marineros sólo son novios de las chicas que son fans de Julio Iglesias**

Solución: Existe algún *x* que es marinero y para todos los individuos *y*, si *x* es novio de *y* entonces los individuos *y* son las chicas fans de Julio Iglesias.

Representación:	M(x)	*x* es marinero
	N(x, y)	*x* es novio de *y*
	JI(x)	*x* es fans de Julio Iglesias
Fórmula:	$\exists x\ [\ M(x) \wedge \forall y\ (\ N(x, y) \rightarrow JI(y)\ )\ ]$	

16.- **Sólo los españoles se dejan conquistar por las chicas de Informática**

Solución: Para todo *x* e *y*, si *x* se deja conquistar por *y* e *y* es un chica de Informática entonces *x* es español.

Representación:	E(x)	*x* es español
	C(x, y)	*x* se deja conquistar por *y*
	I(x)	*x* es chica de Informática
Fórmula:	$\forall x \forall y\ (\ C(x, y) \wedge I(y) \rightarrow E(x)\ )$	

5. TRABAJOS COMPLEMENTARIOS

1.- Estudio de la implicación: implicación material, implicación formal, relación de contenido del antecedente y el consecuente.

2.- Formalización del Lenguaje Natural. Necesidad de un Lenguaje Artificial.

3.- La lógica como herramienta de representación del conocimiento en Inteligencia Artificial.

4.- Introducción al Cálculo de Orden Superior.

6. LA LÓGICA EN LA VIDA

1. Bertrand Russell estaba tratando sobre los enunciados condicionales y sosteniendo que un enunciado falso implica cualquier cosa. Un filósofo escéptico le preguntó:
   - ¿ Quiere usted decir que si 2+2=5, entonces es usted el Papa ?
   Russell contesto afirmativamente y dio la divertida “prueba” que sigue:

- Si suponemos que 2+2=5, entonces seguramente estará usted de acuerdo en que si restamos 2 de cada lado de la ecuación, nos da 2=3. Invirtiendo los términos, tenemos 3=2 y restando 1 de cada lado de la ecuación, nos da 2=1. De modo que, como el Papa y yo somos dos personas, y 2=1, entonces el Papa y yo somos uno. Luego, yo soy el Papa.

2. Esta oración es falsa

¿Qué valor de verdad le darías a la frase anterior? Supongamos que el enunciado anterior sea verdadero, por tanto por lo que dice, la frase es falsa, ¡y nosotros habíamos supuesto que es verdadero! Por otro lado, si el enunciado fuese falso, por lo que dice, la oración no sería falsa, es decir, ¡la frase debería ser verdadera y hemos supuesto que es falsa!

3. Un término puede referirse a muchos objetos y tener un sólo significado, dependiendo de cuándo, dónde y por quién sea pronunciado:

"Dos amigos están hablando del triste estado de la moral sexual de hoy en día:

- Yo nunca me acosté con mi mujer antes de que nos casáramos - dice uno de ellos, haciéndose el santo -. ¿ Y tú ?

- No estoy seguro - responde el otro -. ¿ Cómo se llama ?"

4. En un pueblo se ha formado un club llamado el "Club de los Corazones", que tiene los siguientes estatutos:

a) Dada cualquier mujer del pueblo, si no pertenece a todos los clubs, pertenece al Club de los Corazones.

b) Ningún hombre puede ser socio del Club de los Corazones si no hay al menos otro club del que no es socio.

c) Dado cualquier club, cada hombre ajeno al club ama a cada mujer del Club de Corazones.

Si Chelo y Félix son dos habitantes del pueblo, ¿puede decirse si Félix ama a Chelo?

Capítulo **3**

TEORÍA SEMÁNTICA

Como ya se ha visto, la construcción de un Sistema Formal implica la definición de la noción de verdad para la interpretación de las fórmulas de dicho sistema obteniendo así un Sistema Formal Interpretado. En este capítulo vamos a ver el aspecto semántico de la lógica, tanto para el cálculo de Proposiciones como para el cálculo de Predicados. Como se trata de la lógica clásica, únicamente tendremos dos posibles valores de verdad: verdadero y falso.

Cada enunciado representado en el lenguaje de la lógica, tiene un valor de verdad que tenemos que determinar. Si este enunciado es atómico, el valor de verdad se obtiene directamente por aplicación del conocimiento sobre la información del enunciado a tratar.

Verdad de una Fórmula Atómica: una fórmula atómica es verdadera cuando lo es conforme con los hechos, esto es, cuando la propiedad designada por el predicado corresponde al objeto u objetos individuales de que se trate; en caso contrario el enunciado es falso.

Así por ejemplo el enunciado “Madrid es la capital de España” será verdadero, ya que sabemos que la propiedad de ser capital de España recae en este momento en la ciudad de Madrid. La cuestión de si un enunciado es verdadero o falso no es, como decía Wittgenstein, un problema de análisis lógico, sino un problema de información empírica, porque el enunciado atómico dice siempre algo sobre los hechos, y no es la lógica, sino la experiencia, la que informa sobre la verdad o falsedad de un enunciado de esta índole. Cuando un enunciado sea verdadero, diremos que su valor de verdad es positivo (se representa por V o 1); y cuando sea falso, que tiene valor de verdad negativo (se representa por F o 0). A la verdad o falsedad de los enunciados se les

llamará *valores de verdad*; este concepto de valor de verdad por la que se extiende al orden lógico la terminología usual de la teoría matemática de funciones procede de Peirce y Frege. Trabajaremos de ahora en adelante con enunciados que se tienen que atener al siguiente criterio:

Axioma de la Lógica de Primer Orden: Toda proposición es cierta o falsa, pero no ambas cosas a la vez.

Este criterio también se llama bivalente y se debe a Aristóteles. Este criterio se ha mantenido hasta el presente siglo en que se ha planteado el problema de la no aceptación debido a criterios de la llamada lógica no clásica. Un ejemplo es la Lógica Multivalente que consideran el valor de verdad de los enunciados mayor a dos (verdadero, falso, indeterminado, etc.). Estas lógicas tienen su aplicación en los campos de Física Cuántica y la Informática, entre otros.

1. SEMÁNTICA EN CÁLCULO DE PROPOSICIONES

El sistema de fórmulas o estructuras deductivas correctas no se construye a partir de reglas de inferencia y de axiomas, sino mediante una simbolización del significado de las proposiciones, es decir, de la forma de valorar el contenido de información de cada proposición.

Elementos con los que vamos a trabajar:

- *Conjunto de significados atribuibles a las proposiciones.*
- *Definición semántica de conectivas.*
- *Estudio de tablas de verdad.*
- *Definición semántica de deducción correcta.*

- **Conjunto de significados atribuibles a las proposiciones:**

Los significados con los que vamos a trabajar van a estar comprendidos en el conjunto de valores { *V*, *F* }:

V ..... simboliza el significado verdadero.
F ..... simboliza el significado falso.

Estos valores dan lugar a las mismas fórmulas válidas de la Lógica Clásica o *bivalente*. Dificultades de adaptación al lenguaje natural han dado lugar a la aparición de *lógicas polivalentes* (con tres o más valores de verdad) y *lógicas difusas* y *probabilísticas* (con valores de verdad continuos), que permiten alcanzar un nivel de formalización más adaptado a la semántica del lenguaje usual. De todos modos, como el

objetivo es la formalización del conocimiento científico, la lógica bivalente es válida en la mayoría de los casos. Por otra parte, el estudio de la lógica bivalente nos servirá como base y nos permitirá, en su momento, el paso gradual a otras lógicas con más valores de verdad (trivalente, tetravalente, polivalente, difusa, ...).

- **Definición semántica de conectivas:**

Las distintas conectivas, que se estudiaron en el capítulo referente al Lenguaje de la Lógica de Primer Orden, generarán significados de las frases compuestas. Los distintos valores de verdad de estas frases compuestas dependerán de los valores de verdad de sus componentes y de las conectivas implicadas, interpretados mediante las *tablas de verdad* correspondientes vistas en el capítulo anterior:

A	**B**	**¬A**	**A ∧ B**	**A ∨ B**	**A → B**	**A ↔ B**
V	V	F	V	V	V	V
	F		F	V	F	F
F	V	V	F	V	V	F
	F		F	F	V	V

- **Estudio de tablas de verdad:**

Dada una fbf *A*, cada valoración de verdad, *v*, asigna a *A* un valor *v(A)*. El valor *v(A)* sólo depende de los valores $v(A_i)$ de las variables proposicionales A_i contenidas en *A*, y por tanto es finito, del orden de 2^n, donde *n* es el número de variables distintas contenidas en dicha fórmula. Vamos a ver dos métodos para calcular el valor de verdad de una fbf mediante tablas de verdad, pero antes es conveniente saber construir una tabla de verdad. Para ello, crearemos un cuadro de doble entrada con tantas filas y columnas como describimos a continuación:

Columnas: dependerá del método utilizado; en principio se pone una columna por cada variable que aparezca en la fórmula.

Filas: si en la fórmula aparecen *n* variables tenemos 2^n filas que se construyen de la siguiente forma:

- 1ª var.: $\frac{2^n}{2}$ valores V seguidos de $\frac{2^n}{2}$ valores de F; hasta 2^n.
- 2ª var.: $\frac{2^n}{2^2}$ valores V seguidos de $\frac{2^n}{2^2}$ valores de F; hasta 2^n.

...

- n^{ava} var.: 1 valor V seguidos de 1 valor de F; hasta 2^n.

Vamos a ver los dos métodos por medio de un ejemplo: construir la tabla de verdad para la fórmula $\neg p \vee q$.

1.- Método Acumulativo:

Ampliamos el número de columnas en:

- una columna nueva por cada variable que aparece negada.
- una columna nueva por cada operación básica que haya que realizar con las conectivas.

Este método se aplica a fórmulas no demasiado complicadas como $\neg p \vee q$ o similares. Seguimos los siguientes pasos:

- Se escriben las dos 1ª columnas con todos los posibles valores de verdad para las variables proposicionales *p* y *q*.
- Se analiza $\neg p$
- Se analiza $\neg p \vee q$

Tabla:

p	q	¬p	¬p ∨ q
V	V	F	V
V	F	F	F
F	V	V	V
F	F	V	V

2.- Método por Pasos:

Tendremos tantas columnas como variables distintas, más una columna adicional para la fórmula global. En el ejemplo que seguimos será:

- Como hay dos variables se disponen las dos primeras columnas con todos los valores posibles de *p* y *q*.
- Se escribe en cabecera de tabla la fórmula a analizar, procurando separar convenientemente cada uno de sus símbolos.
- Se dispone a pie de tabla un casillero para registrar el orden de los pasos a dar.
- Como en el método anterior, se realizan los análisis parciales de la fórmula, siendo el primer paso, la transcripción de los valores de verdad de las variables proposicionales.
- La columna que tenga el número de paso más elevado registrará el valor de verdad de la fórmula analizada.

Tabla:

p	q	¬p ∨ q
V	V	F V V
V	F	F F F
F	V	V V V
F	F	V V F
		2 3 1

Una vez construida la *tabla de verdad* nos fijamos en la columna solución de la tabla y según los resultados podemos determinar el valor de verdad de la fórmula de acuerdo con las siguientes definiciones:

Tautología : una fbf del Cálculo de Proposiciones se dice que es Tautología o Identidad Lógica, cuando es verdadera para toda interpretación; es decir, cuando toda atribución veritativa la satisface.

Ejemplo: $p \vee \neg p$

Contradicción : una fbf del Cálculo de Proposiciones se dice que es Contradicción cuando no es verdadera bajo ninguna interpretación; es decir, cuando ninguna atribución veritativa la satisface.

Ejemplo: $p \wedge \neg p$

Contingencia : una fbf del Cálculo de Proposiciones se dice que es Contingencia o Indeterminación cuando no es ni tautología ni contradicción; es decir, cuando existe al menos una atribución veritativa que la satisfaga y otra que no la satisfaga.

Ejemplo: $p \vee q$

Se dice que una valoración de verdad *v* **satisface** a una fbf *A* cuando *v(A)=V*. Sea *W* un conjunto de fbf y *A* una fbf, diremos que *W* **implica tautológicamente** a *A* si cada valoración de verdad que satisface a todas las fbf de *W*, satisface también a *A*. Entonces se escribe $W \models A$; y en caso contrario se escribe $W \not\models A$.

Ejemplos:

$\{ A, A \rightarrow B \} \models B$
$\{ A \vee B, \neg B \} \models A$
$\{ A \vee B \} \not\models B$

ACLARACIONES:

1.- Se admite que cualquier valoración de verdad satisface a las fbf del conjunto de fbf vacío, $W=\varnothing$. Esto tienen sentido ya que no existe ninguna fbf del vacío que sea falsa en alguna valoración. En tal caso se dice que *A* es una fbf tautológica o que es una tautología. Se indica escribiendo $\models A$. Lo mismo sucede si todas las fórmulas de *W* son tautológicas pues entonces cualquier valoración de verdad satisface a todas las fbf de *W*.

2.- Si no existe ninguna valoración de verdad que satisfaga a todas las fbf de *W*, se tiene que $W \models A$ para cualquier fbf *A*, trivialmente. En efecto por definición de $\models$ solamente no se cumple $W \models A$ en el caso de que exista una valoración que haga verdad a todas las fbf de *W* y haga falsa a *A*, y esto no puede ocurrir ahora.

Ejemplos:

$$\models \quad A \rightarrow (B \rightarrow A)$$

$$\{ A, \neg A \} \models C \qquad \text{(para cualquier C)}$$

Se dice que dos fbf *A* y *B* son **tautológicamente equivalentes** si $A \models B$ y $B \models A$. Se escribe entonces $A \Leftrightarrow B$.

NOTA: Dos proposiciones en el lenguaje ordinario, cuya representación en Lógica Proposicional sean dos fbf tautológicamente equivalentes expresan el mismo hecho, ya que una es verdad si y sólo si es verdad la otra. Esto es, desde el punto de vista de la Lógica ordinaria, lo mismo da enunciar la una que la otra. Se suele decir que son semánticamente equivalentes.

Métodos de Verificación de Tautologías:

1.- *Mediante tablas de verdad*: se construye la tabla de verdad con alguno de los métodos vistos, y entonces puede suceder:

- Si en la columna resultado encontramos todos los valores a V entonces la fbf es una **tautología.**
- Si en la columna resultado encontramos todos los valores a F entonces es una **contradicción.**
- Cuando alternan símbolos V y F entonces la fbf es una **contingencia.**

Ejemplo:

Estudiar el valor de verdad de la siguiente fbf por el método de las tablas de verdad.

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

Aplicando el método acumulativo obtenemos la siguiente tabla:

p	q	¬p	p ∨ q	¬p → q	(p∨q) ∧ (¬p→q)
V	V	F	V	V	V
V	F	F	V	V	V
F	V	V	V	V	V
F	F	V	F	F	F

La columna solución (la última) nos revela que la fbf es una «contingencia», pues aparecen tanto valores V como valores F.

2.- *Método corto de valoración*: consiste en dar el valor de verdad F a la fórmula *A*, y si de esta hipótesis se deduce que hay una contradicción entonces *A* es una tautología. Si no aparece la contradicción, entonces suele obtenerse una valoración en la que *A* es falsa y por tanto no es tautología. El valor supuesto de *A* se escribe debajo de su conectiva principal, la que define el tipo.

Ejemplo:

$$[\,(p \wedge q) \rightarrow r\,] \rightarrow [\,p \rightarrow (q \rightarrow r)\,]$$
$$\qquad\qquad\qquad\quad F$$
$$\qquad\quad V \qquad\qquad F$$
$$\qquad\qquad\quad V \qquad F$$
$$\qquad\qquad\qquad\quad V \quad F$$
$$V \quad V \qquad F$$

Hemos llegado a una contradicción, pues la implicación primera es V y sin embargo r a la fuerza es F y el antecedente (p ∧ q) es V. Luego, la hipótesis de que partimos, implicación falsa, es incorrecta, por tanto es una fórmula tautológica, ya que no podemos conseguir valores de verdad que la hagan falsa.

Propiedades de las Tautologías:

1. Las tautologías constituyen un conjunto de enunciados que es **decidible.** Un conjunto es decidible cuando existe un procedimiento mecánico para determinar cuales son los elementos que le pertenecen y los que no le pertenecen. En las tautologías basta aplicar cualquiera de los métodos vistos para decidir si una fbf es una tautología o no.

2. **Sustitutividad.** Dada una tautología si se sustituye en ella una letra enunciativa (en todas sus ocurrencias) por una fórmula cualquiera el resultado es una fórmula equivalente.

   Ejemplo:
   Sea $p \rightarrow p$ que sabemos que es una tautología, si sustituimos la variable *p* por $p \wedge q$, queda una fórmula $p \wedge q \rightarrow p \wedge q$ que también es una tautología.

3. Las equivalencias tautológicas se sujetan a la **ley de intercambio.** Dadas dos fórmulas equivalentes *A* y *B* y dada una fórmula *C* que contiene como subfórmula la

ocurrencia de una de ellas, puede cambiarse en *C* dicha ocurrencia de esa subfórmula por la ocurrencia de su equivalente. Si una aplicación de esa regla afecta a una ocurrencia que aparece más veces, hay que indicarlo.

Ejemplo:
Sabemos que p y $\neg\neg p$ son equivalentes. Por tanto si tenemos la fórmula $q \rightarrow (\neg\neg p \vee r)$, podemos sustituir obteniendo la fórmula $q \rightarrow (p \vee r)$, que tendrá el mismo valor de verdad.

- **Definición semántica de Deducción Correcta:**

Veamos algunos conceptos pertenecientes a la categoría semántica de la lógica:

Interpretación : de una fórmula es una asignación de significados a sus fórmulas componentes básicas. Una interpretación correspondería a una fila de la tabla de verdad.

Modelo : una interpretación es un modelo de una fórmula si el significado resultante en la misma es *verdad*. Dado un conjunto de fórmulas, una interpretación es un modelo del mismo si satisface todas sus fórmulas.

Contramodelo o **contraejemplo** : es una interpretación cuyo significado resultante es *falso*.

Fórmula **Semánticamente Válida** : es aquella que no tiene contraejemplos, es decir, todas las interpretaciones son modelos, o lo que es lo mismo, tienen valor *verdadero*. Las tautologías son fórmulas semánticamente válidas.

Concepto semántico de deducción:

Dada una estructura deductiva $P_1, P_2, ..., P_n \Rightarrow Q$ se define como **CORRECTA** cuando no existe ninguna interpretación que simultáneamente haga $P_1, P_2, ... P_n$ verdaderos y Q falso, es decir, cuando todo modelo de las *premisas* es un modelo de la *conclusión*.

Ejemplo: Comprobemos con esta definición que la estructura del MP es correcta:

$A, A \rightarrow B \Rightarrow B$

	A	**B**	**A → B**
1	V	V	V
2	V	F	F
3	F	V	V
4	F	F	V

Según la definición anterior la única fila que hace verdaderas las premisas (todas) es la fila 1, y aquí se comprueba que la conclusión es verdadera; luego es una deducción correcta. Las demás filas no nos dicen nada al tener premisas falsas.

La definición semántica de la deducción es la base del *método del contramodelo* o *contraejemplo*: para comprobar que una deducción no es correcta basta encontrar una interpretación que atribuya significado verdadero a las premisas y falso a la conclusión.

Ejemplo:

Comprobar por el método del contraejemplo si la siguiente deducción es correcta:

$$p \vee q\ ,\ q \rightarrow r\ ,\ p \rightarrow s\ \Rightarrow\ s$$

Hay que encontrar una interpretación que haga verdaderas las premisas y falsa la conclusión.

Hacemos la conclusión falsa: $s \leftarrow$ F

Si s = F, para que $p \rightarrow s$ sea V tenemos que hacer $p \leftarrow$ F

Si p = F, para que $p \vee q$ sea V tenemos que hacer $q \leftarrow$ V

Si q = V, para que $q \rightarrow r$ sea V tenemos que hacer $r \leftarrow$ V

Luego los valores de verdad $p \leftarrow$ F
$q \leftarrow$ V
$r \leftarrow$ V
$s \leftarrow$ F

hacen que todas las premisas sean verdaderas y la conclusión falsa, entonces tenemos claramente un contraejemplo. Luego *el argumento no es correcto*.

2. SEMÁNTICA EN CÁLCULO DE PREDICADOS

De una fórmula en cálculo de proposiciones decimos que es verdadera o falsa según los valores de verdad que se atribuya a sus componentes atómicas. Con las fórmulas cuantificadas, se dice lo mismo pero teniendo en cuenta que las componentes atómicas no están de manifiesto, ni estarán, hasta que no se especifique la cardinalidad del universo elegido.

Ejemplo:

$\forall x\ P(x)$ no es una fórmula atómica sino molecular. Sus componentes atómicas son predicados del tipo $P(a)$ y serán tantos como el número de individuos tenga el Universo que sirva de rango a la variable x. O sea, si suponemos que hay dos individuos, tenemos $D = \{ a, b \}$, entonces hay dos predicados, $P(a)$ y $P(b)$, que serían ya fórmulas atómicas.

$$\forall x\ P(x) \Leftrightarrow P(a) \wedge P(b)$$

Y ya podemos construir la *tabla de verdad* para está fórmula:

P(a)	P(b)	P(a) ∧ P(b)
V	V	V
V	F	F
F	V	F
F	F	F

Por tanto, mientras el número de individuos del Universo sea finito, podemos hacer la tabla de verdad para cualquier expresión cuantificada, aunque cuando haya muchos elementos sea materialmente imposible realizarla. Por ello hay que indicar, no sólo que el Universo es finito, sino también cuantos individuos lo componen.

Validez de Fórmulas Cuantificadas

Para demostrar la validez de una fórmula cuantificada en Tª Semántica, tenemos que definir una **interpretación** de dicha fórmula, que requiere tener en cuenta las siguientes componentes:

- **Universo o Dominio de Referencia**:

Dada una fórmula se elige arbitrariamente como zona de referencia del mismo a un conjunto o colección de objetos cualquiera *D*, en el cual se van a establecer relaciones y propiedades de individuos, que marcarán subconjuntos de dicho conjunto. Dicho conjunto tiene que ser:

- no vacío
- individuos distinguibles entre sí

En una fórmula cuantificada los individuos que pueden aparecer son:

- *constantes*: un elemento concreto del dominio
- *variables*: un elemento cualquiera del dominio
- *funciones*: un elemento definido en función de otro u otros (*n*=número de argumentos de la función):

$$f : D^n \rightarrow D$$

Ejemplo:

Si en una fórmula lógica aparece la función *f(x)* y estamos trabajando en el dominio *D={a, b, c}*, una posible interpretación de la función *f* sería la definida mediante $D \rightarrow D$ representada en la tabla siguiente:

x	f(x)
a	b
b	c
c	b

- **Conjunto de Significados** que se atribuyen a las fórmulas. Al igual que en el cálculo de proposiciones tendremos:

  V.....Verdadero
  F..... Falso

- **Relación *n*-ária**, siendo *n* la aridad del predicado, entre elementos del dominio correspondiente a cada letra de *predicado*:

$$P : D^n \rightarrow \{ V, F \}$$

Ejemplo:

Si en una fórmula lógica aparece el predicado *P(x)* y estamos trabajando en el dominio *D={a,b,c}*, una posible interpretación de dicho predicado *P* podría ser la definida mediante *D→{V,F}* representada en la tabla siguiente:

x	P(x)
a	V
b	F
c	V

Se representa por una correspondencia entre el conjunto de todas las *n*-tuplas elementos del dominio *D* (D^n) y el conjunto de significados {*V*, *F*} asignando a cada *n*-tupla de D^n que verifica la relación el significado *V*, y *F* en caso contrario. Así, tendremos d^n posibles combinaciones de valores, donde *d* es el número de elementos del dominio *D* y *n* es la aridad del predicado *P*. También se puede comprobar que el número de interpretaciones posibles para cada predicado es del orden de:

$$2^{(d^n)}$$

Ejemplo:

Si en una fórmula lógica aparece el predicado *P(x, y)* y estamos trabajando en el dominio *D = {a, b, c}*, una posible interpretación del predicado *P* sería la definida mediante $D^2 \rightarrow \{V, F\}$ representada en la tabla siguiente. Como *n*=2 (aridad del predicado *P*) y *d*=3 (nº elementos del dominio *D*), tendremos 3^2=9 filas en la tabla.

x	y	P(x, y)
a	a	V
	b	V
	c	F
b	a	V
	b	V
	c	F
c	a	V
	b	V
	c	F

Como ya hemos dicho, esta es una posible interpretación del predicado *P*. Pero, ¿de cuántas maneras distintas podemos interpretar a *P*? Como para definir *P* debemos asignar 2 posibles valores de verdad (*V* o *F*) a 9 combinaciones de elementos del dominio, tendremos en total 2^9=512 interpretaciones posibles.

- Definición semántica de **Conectivas**: *Tablas de Verdad*.

Aquí sirve lo que se ha dicho anteriormente para el cálculo de proposiciones, ya que las conectivas utilizadas son las mismas.

- Definición semántica de **Cuantificadores**:

a) Una fórmula cuantificada universalmente respecto a *x* es *V* si para cualquier (todo) elemento del dominio asignado a *x* la fórmula es verdadera. *F* en otro caso.

b) Una fórmula cuantificada existencialmente respecto de *x* es *V* si para algún (al menos uno) elemento del dominio asignado a *x* la fórmula es *V*. *F* si todos son falsos.

Ejemplo:

Sean la siguiente interpretación para los predicados *P(x, y)* y *Q*(*x*) en el dominio *D={a,b,c}*:

x	y	P(x, y)	Q(x)
a	a	F	V
	b	V	
	c	V	
b	a	V	V
	b	V	
	c	V	
c	a	F	F
	b	F	
	c	V	

Las fórmulas lógicas siguientes toman el valor de verdad:

a)	∀x Q(x)	↵	F	
b)	∃y Q(y)	↵	V	
c)	∀y P(b,y)	↵	V	
d)	∀x ∃y P(x,y)	↵	V	
e)	∀x ∀y P(x,y)	↵	F	
f)	∃x ∀y P(x,y)	↵	V	(sí, cuando x=b)
g)	∃y ∀x P(x,y)	↵	V	(sí, cuando y=c)

- Definición semántica **Deducción Correcta**.

Aunque sirve lo comentado en el cálculo de proposiciones, vamos a matizar alguno de los conceptos implicados:

Satisfacible : una fórmula es satisfacible si existe alguna interpretación que la verifique.

Validez : una fórmula es válida en un dominio si cualquier interpretación que pueda plantearse en ese dominio satisface la fórmula.

Validez Semántica : una fórmula es semánticamente válida cuando es válida en cualquier dominio.

Ejemplo:

$$\exists x \forall y P(x, y) \rightarrow \forall y \exists x P(x, y)$$

es verdad lógica, porque no admite ninguna interpretación que la haga falsa.

Modelo : es una interpretación que satisface una fórmula o un conjunto de fórmulas.

Contramodelo o **Contraejemplo** : interpretación que no satisface una fórmula o conjunto de fórmulas.

Ejemplo:

Si observamos el ejemplo anterior (apartado de "definición semántica de Cuantificadores") vemos que la interpretación dada en la tabla:

- es un *modelo* de las fórmulas b, c, d, f y g
- es un *contramodelo* de las fórmulas a y e

Equivalencias Lógicas

E1)	$p \wedge \neg p \equiv F$	E2)	$p \vee \neg p \equiv V$
E3)	$p \wedge V \equiv p$	E4)	$p \vee V \equiv V$
E5)	$p \wedge F \equiv F$	E6)	$p \vee F \equiv p$

3. ESTUDIO Y ANÁLISIS DEL CONJUNTO DE CONECTIVAS

Hemos visto en nuestro lenguaje cinco conectivas lógicas ($\neg$, $\wedge$, $\vee$, $\rightarrow$ y $\leftrightarrow$) pero de hecho podíamos haber definido más conectivas, e incluso prescindo de algunas de ellas. Vamos, por tanto, ahora a estudiar y analizar el porqué hemos elegido únicamente estas cinco y cuáles más podíamos haber definido.

Si calculamos todas las posibles conectivas binarias que se pueden definir con los dos valores de verdad (V y F) obtenemos que, como se trata de conectivas binarias (con dos argumentos), tenemos 4 combinaciones (2^2) y como a cada combinación le podemos asignar cualquier de los dos valores de verdad, tenemos $2^4 = 16$ diferentes definiciones de conectivas binarias.

Ejemplo:

A	B	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}
V	V	V	V	V	V	V	V	V	V	F	F	F	F	F	F	F	F
	F	V	V	V	V	F	F	F	F	V	V	V	V	F	F	F	F
F	V	V	V	F	F	V	V	F	F	V	V	F	F	V	V	F	F
	F	V	F	V	F	V	F	V	F	V	F	V	F	V	F	V	F

Se puede ver que la c_2 corresponde a la disyunción, la c_5 corresponde a la implicación, la c_7 corresponde a la coimplicación y la c_8 corresponde a la conjunción

Para las conectivas unarias tenemos 2 combinaciones (2^1) con dos posibles valores de verdad para cada una de ellas, que hacen un total de $2^2 = 4$ conectivas distintas.

Ejemplo:

A	n_1	n_2	n_3	n_4
V	V	V	F	F
F	V	F	V	F

Podemos observar fácilmente que:

- la conectiva n_1 hace siempre verdadera la fórmula sea cual sea su valor
- la conectiva n_2 mantiene el valor original de la fórmula
- la conectiva n_3 es la negación que hemos definido antes (cambia el valor de verdad)
- la conectiva n_4 hace siempre falsa la fórmula sea cual sea su valor

Vemos que tenemos 16+4=20 conectivas distintas. Pero no todas ellas son imprescindibles, es decir, con un subconjunto de ellas podemos simular el efecto de las

otras. Si lo estudiamos detenidamente podemos comprobar que con las conectivas $\neg$, $\wedge$ y $\vee$ podemos representar cualquier tabla de verdad para las conectivas anteriormente descritas, es decir, $\{\neg, \wedge, \vee\}$ es un conjunto *funcionalmente completo* de conectivas (completo desde el punto de vista del valor de verdad).

Ejemplo:

La conectiva c_{10} corresponde a la disyunción exclusiva. Esta conectiva la podemos representar, en función de $\{\neg, \wedge, \vee\}$, como

$$(A \vee B) \wedge \neg(A \wedge B)$$

A	B	c_{10}	$A \wedge B$	$\neg(A \wedge B)$	$A \vee B$	$(A \vee B) \wedge \neg(A \wedge B)$
V	V	F	V	F	V	F
	F	V	F	V	V	V
F	V	V	F	V	V	V
	F	F	F	V	F	F

Podemos hacernos la pregunta de ¿por qué hemos utilizados hasta el momento cinco conectivas?, ¿qué nos aportan las conectivas $\rightarrow$ y $\leftrightarrow$? Porque estas conectivas incrementan nuestra potencia expresiva y simplifican notablemente nuestras fórmulas. La implicación nos permite simbolizar las sentencias condicionales tan usadas en lenguaje natural y el coimplicador simboliza las relaciones de equivalencia.

4. EJERCICIOS

1.- Demostrar que la fórmula siguiente es satisfacible para la interpretación dada.

$$\forall x\,[\,P(x) \rightarrow R(x)\,] \rightarrow \exists y\,Q(y)$$

Interpretación:

Dominio: $D = \{a, b, c\}$

Predicados:

x	P(x)	R(x)	Q(x)
a	V	F	F
b	V	V	F
c	F	F	F

Solución:

Para esta interpretación dada, calculamos el valor de verdad de la fórmula:

x	P(x)	R(x)	$P(x) \rightarrow R(x)$	$\forall x\,(P(x) \rightarrow R(x))$
a	V	F	F	F
b	V	V	V	
c	F	F	V	

Por otro lado, como *Q(y)* es *F* para todo valor de *y* entonces *∃ y Q(y)* es *F*. Luego nos queda al final:

$$F \rightarrow F$$

que es verdad siempre, por lo que la fórmula es *satisfacible.*

2.- Demostrar que, para la interpretación dada, la siguiente fórmula es satisfacible

$$\mathbf{P(x) \wedge \forall x\,[\,R(x) \wedge P(y)\,] \rightarrow \exists y\, Q(x, y)}$$

Interpretación:

Dominio: D = { 1, 2 }

Términos: x = 1, y = 2 para las *x* e *y* que son libres.

Predicados:

x	y	Q(x, y)	R(x)	P(x)
1	1	V	F	V
	2	F		
2	1	V	V	F
	2	F		

Solución:

Interpretando las variables libres la fórmula queda:

$$P(1) \wedge \forall x\,[\,R(x) \wedge P(2)\,] \rightarrow \exists y\, Q(1, y)$$

x	R(x)	R(x) ∧ P(2)	∀x [R(x) → P(2)]	P(1) ∧ ∀x [R(x)→P(x)]
1	F	F	F	F
2	V	F		

y	Q(1, y)	∃y Q(1, y)
1	V	V
2	F	

Luego, como F → V es verdadero, la fórmula es *satisfacible.*

3.- Sea la siguiente fórmula bien formada del Cálculo de Predicados:

$$\mathbf{\forall x\,[\,(P(x) \rightarrow q) \wedge \exists y\, Q(x, y)\,]}$$

a) ¿Cuántas interpretaciones posibles tiene cada uno de los predicados que aparecen en la fórmula en el dominio *D* = {a, b, c}? ¿Y en total para la fórmula completa?

Solución:

Predicado *P* : d=3 elementos en el dominio *D*={a, b, c}
n=1 aridad del predicado *P*
2 valores de verdad {V, F}

$$2^{(3^1)} = 8 \text{ interpretaciones}$$

Predicado *Q* : d=3 elementos en el dominio *D*={a, b, c}
n=2 aridad del predicado *Q*
2 valores de verdad {V, F}

$2^{(3^2)} = 2^9 = 512$ interpretaciones

Variable proposicional *p* : 2 valores de verdad {V, F}

2 interpretaciones

La fórmula tendrá:

8 x 512 x 2 =8192 interpretaciones

b) Dada la siguiente interpretación para el dominio D = {a, b}, estudiar por tablas de verdad cuál es la valoración semántica de la fórmula.

x	y	P(x)	Q(x, y)
a	a	V	V
	b		V
b	a	F	F
	b		V

Solución:

Tendremos dos posibles interpretaciones según el valor que tome la variable proposicional *q*

q	x	y	P(x)	P(x) → q	Q(x, y)	∃y Q(x, y)	(5) ∧ (7)	∀x (8)
V	a	a	V	V	V	V	V	V
		b			V			
	b	a	F	V	F	V	V	
		b			V			
F	a	a	V	F	V	V	F	F
		b			V			
	b	a	F	V	F	V	V	
		b			V			

Según los resultados obtenidos en la tabla de verdad, está interpretación, cuando *q* es *F*, es un **contramodelo** de la fórmula en este dominio; y es un **modelo** cuando *q* toma el valor verdadero.

4.- Formalizar en el Cálculo de Proposiciones y buscar un contraejemplo del siguiente argumento:

- **Si tomo un café y un donuts, trabajaré más feliz**.
- **Si trabajo más feliz y me visitan los amigos entonces no acabaré la carta**.

- Si hoy es viernes, me visitarán los amigos y terminaré la carta.
***Luego,* si tomo un café y me visitan los amigos, hoy es viernes**.

Solución:

Formalización:

c	tomar café	
d	tomar donuts	$- c \wedge d \rightarrow f$
f	trabajar feliz	$- f \wedge v \rightarrow \neg a$
v	visitan los amigos	$- h \rightarrow v \wedge a$
a	acabar la carta	$\Rightarrow c \wedge v \rightarrow h$
h	hoy es viernes	

Contraejemplo: Se trata de encontrar una interpretación que haga falso el argumento, es decir, que sean verdaderas las premisas y falsa la conclusión.

$c \wedge d \rightarrow f$ ↯ Verdadero
$f \wedge v \rightarrow \neg a$ ↯ Verdadero
$h \rightarrow v \wedge a$ ↯ Verdadero
$c \wedge v \rightarrow h$ ↯ Falso por tanto $c \wedge v$ ↯ V (**c ↯ V** y **v ↯ V**) y **h ↯ F**
como h ↯ F y $h \rightarrow v \wedge a$ ↯ V, $v \wedge a$ puede ser V o F. Hagamos $v \wedge a$ ↯ V (**a ↯ V**)
como a ↯ V y por tanto ¬a ↯ F, y $f \wedge v \rightarrow \neg a$ ↯ V, $f \wedge v$ ↯ F. Como v ↯ V, **f ↯ F**
como f ↯ F y $c \wedge d \rightarrow f$ ↯ V, entonces $c \wedge d$ ↯ F, y por tanto como c ↯ V, **d ↯ F**

La siguiente interpretación: **c ↯ V** , **d ↯ F** , **f ↯ F** , **v ↯ V** , **a ↯ V** y **h ↯ F** hace verdaderas las 4 premisas y falsa la conclusión. Por tanto el argumento no es correcto.

5. TRABAJOS COMPLEMENTARIOS

1.- Estudio de lógicas no bivalentes:
- Lógica trivalente (*verdadero, falso, indeterminado*)
- Lógica Difusa ("Fuzzy Logic"): función de *posibilidad*
- Lógica Probabilística: función de *probabilidad*

2.- Realización de un programa de ordenador que construya la "tabla de verdad" de una fórmula dada del Cálculo de Proposiciones.

6. LA LÓGICA EN LA VIDA

1. Jordi va a la agencia matrimonial por ordenador «Que usted lo pase bien» para indicar sus requisitos (axiomas). Quiere que sea blanca, no muy charlatana, que se encuentre cómoda entre pieles pero que, al mismo tiempo, desdeñe la vida de la ciudad. El ordenador le manda una osa polar.

2. Un principio básico de la lógica clásica es la *ley de exclusión del término medio*, que formalizada sería $A \vee \neg A$.

Había un profesor de lógica tan "lógico" que podía analizar cualquier situación por compleja que fuera. Sus alumnos se preguntaban si su capacidad de razonamiento podría resistir unas rondas de vino. Así que, durante una fiesta, esos curiosos alumnos le indujeron a beber bastante vino como para achisparle. Cuando se durmió, le llevaron al cementerio y le echaron al suelo detrás de una losa. Después se escondieron para esperar su análisis de la situación. Cuando se despertó, hizo un uso impresionante de la ley de exclusión del término medio:

- O estoy vivo, o no lo estoy. Si estoy vivo, ¿qué hago aquí? Y si estoy muerto, ¿por qué tengo ganas de ir al cuarto de baño?

3. ¿ Qué número, expresado en letras, hay que poner para que la proposición resultante sea VERDADERA ?

«Esta frase tiene _____ vocales»

¿ Qué número, expresado en letras, hay que poner en el hueco para que la proposición resulte FALSA ?

«Esta frase no tiene _____ letras»

4. *Receta de inmortalidad de Raymond Smullyan*:

Para ser inmortal, todo lo que hay que hacer son dos cosas:

- Siempre decir la verdad; nunca hacer ninguna declaración falsa en el futuro.
- Decir simplemente: "¡Repetiré esta oración mañana!"

Si haces estas dos cosas, ¡te garantizo que vivirás para siempre!

Capítulo **4**

DEDUCCIÓN NATURAL EN LÓGICA DE PRIMER ORDEN

Todo Sistema Formal necesita de un método de cálculo que nos permita deducir nuevas fórmulas a partir de las conocidas por simple manipulación sintáctica. Vamos ver dos métodos distintos: la Deducción Natural y el Sistema Axiomático. En este tema nos ocuparemos de la Deducción Natural, que está más cercano al razonamiento intuitivo del ser humano. De forma sencilla, a partir de las fórmulas dadas, y mediante la aplicación de reglas, obtendremos nuevas fórmula. Podemos considerar la deducción como una forma de computación, ya que ¿un programa no es una deducción en la cual a partir de unas entradas (premisas) debemos obtener unas salidas determinadas (conclusiones)?

Conocida la forma (estructura) de nuestras frases (fórmulas bien formadas) y teniendo ya a nuestro alcance la simbolización necesaria, podemos dirigirnos hacia una parte importante de la Lógica Formal que es la Deducción o Inferencia. Las reglas de inferencia que rigen el uso de los términos de enlace son muy sencillas y se aprende como si fuera un juego. El juego se realiza con fbf; la idea del juego es partir de un conjunto de fórmulas que se llaman premisas y obtener otras fórmulas, que se llaman conclusiones, por medio de la aplicación de estas reglas de inferencia.

Deducción : paso lógico de las premisas a la conclusión. La idea de inferencia se puede expresar de la manera siguiente:

« *de premisas verdaderas se obtienen sólo conclusiones que son verdaderas* »

es decir, si las premisas son verdaderas, entonces las conclusiones que se derivan de ellas lógicamente han de ser verdaderas. En el caso de que alguna premisa sea falsa, no podemos afirmar con rotundidad nada de la conclusión.

Ejemplo:

Vamos a dar una idea de como trabajar con inferencias, antes de pasar a las reglas. Se supone que se tienen dos premisas, la fórmula $P \rightarrow Q$ y la fórmula P. Se empieza diciendo que estas premisas están dadas, es decir, se ha dado P y se ha dado $P \rightarrow Q$. Preguntamos ¿se puede sacar una conclusión de estas dos proposiciones?, o lo que es lo mismo, ¿se puede idear otra fórmula que haya de ser cierta si las premisas son ciertas?.

Podemos ponerlo así: Es cierto *si P entonces Q* y, es cierto *P*.

La 1ª fbf expresa que si se verifica P entonces se verifica Q y la 2ª dice que se verifica P, luego es trivial ver que se verifica Q. Se dice que la fbf Q es consecuencia lógica de las premisas P y $P \rightarrow Q$.

1. CONCEPTOS BÁSICOS

Los argumentos están formados por fórmulas lógicas. La valoración de los argumentos no es verdadero o falso, sino bien construidos (correctos) o mal construidos (incorrectos).

Argumento Correcto o **Válido** : es un conjunto de enunciados tal que no es posible que las premisas sean verdaderas y la conclusión sea falsa, es decir, la verdad de las premisas es incompatible con la falsedad de la conclusión.

Esta definición no excluye la posibilidad de argumentos que tengan una o más premisas falsas y conclusión falsa o verdadera y que, sin embargo, sean correctos.

Ejemplo:

La Luna es mayor que el Sol.
El Sol es mayor que la Tierra.
Luego, la Luna es mayor que la Tierra.

La Luna es mayor que el Sol	*p*	(es una premisa falsa)
El Sol es mayor que la Tierra	*q*	(es una premisa verdadera)
La luna es mayor que la Tierra	*r*	(la conclusión es falsa)

Y, sin embargo, el argumento está bien construido.

El método tradicional de exponer los argumentos consiste en aducir 1º las premisas y después la conclusión que está ligada a ellas por la partícula *luego, por tanto*, etc.

Ejemplo:

Si suben los salarios, entonces suben los precios.
Si suben los precios, entonces baja el poder adquisitivo de la moneda.
Suben los salarios.
Luego, baja el poder adquisitivo de la moneda.

Haciendo uso del lenguaje formal que conocemos, este argumento se podría escribir así:

p subida de salarios.
q subida de precios.
r baja el poder adquisitivo de la moneda.

Formalización del argumento:

$$p \rightarrow q,\ q \rightarrow r,\ p \Rightarrow r$$

Al símbolo $\Rightarrow$ le llamaremos DEDUCTOR, que será un símbolo metalingüístico representativo de la relación formal de deducción. La interpretación de éste símbolo, en el ejemplo anterior, indica que *r* se deduce de las premisas $p \rightarrow q,\ q \rightarrow r$ y p.

NOTA: Por metalingüístico entendemos un símbolo que pertenece al lenguaje llamado *Metalenguaje*. Este lenguaje está entre el lenguaje ordinario y el lenguaje objeto y contiene símbolos que son necesarios para la aplicación de nuestro cálculo. Podemos decir que el metalenguaje es el lenguaje castellano acompañado de signos auxiliares y abreviaturas.

1.1. Cálculo Lógico

La tarea de una deducción no se limita a formular argumentos, también es tarea de la Lógica el estudio y la formulación explícita y rigurosa de las reglas que gobiernan las operaciones deductivas. Para formular reglas de inferencia vamos a utilizar variables de fórmulas y símbolos lógicos y vamos a utilizar el símbolo del deductor para separar las premisas de la conclusión. También se suele utilizar una raya horizontal para separar las premisas de la conclusión. En toda regla el orden de las premisas es indiferente.

Ejemplo:

Modus ponens: $A \rightarrow B,\ A \Rightarrow B$ $\qquad$ $\dfrac{A \Rightarrow B \quad A}{B}$

A un conjunto de reglas sistemáticamente ordenado lo llamaremos **cálculo lógico**. Esta analogía se toma por los cálculos matemáticos, cuyas operaciones se efectúan de acuerdo a un conjunto de reglas y sobre la base de un código de símbolos y sistemas de notación adecuados. Similarmente las operaciones lógicas de deducción se efectúan de acuerdo con un conjunto de símbolos y reglas de formación de fórmulas, al que se añade un sistema de reglas de inferencia. Las reglas de inferencia de la Lógica regulan las transformaciones de fórmulas compuestas a base de conectivas y cuantificadores.

Deducción Formal o Derivación : es una secuencia finita de fórmulas tales que cada una de ellas sea:

1.- Supuesto inicial o premisa.
2.- Supuesto provisional
3.- Una fórmula que se derive lógicamente de otra o de otras por inferencia inmediata.

A cada fórmula de la secuencia se le da el nombre de **línea de derivación**. La última línea de derivación se le llama *conclusión final*, y todas las líneas de las que partimos se les llama *premisas.*

1.- Supuestos iniciales o premisas:

Son fórmulas que se consideran hipotéticamente dadas desde el principio de la derivación. Los argumentos pueden tener un número finito cualquiera de premisas. También hay derivaciones que están exentas de premisas como es el caso de las demostraciones.

2.- Supuestos provisionales:

Son líneas que se introducen provisionalmente en el transcurso de la prueba y deberán ser canceladas antes del establecimiento de la conclusión. Es importante que se entienda bien el proceso de cancelación para que pueda ser usado adecuadamente, ya que si no tenemos cuidado podemos demostrar conclusiones que no se siguen de las premisas dadas. Desde la introducción del supuesto provisional hasta la línea en que lo cancelamos estamos trabajando en una *subderivación*, *subdeducción* o *subprueba*.

3.- Líneas que proceden de otra(s) anterior(es) por aplicación de una regla de inferencia:

A estas líneas las llamamos consecuencias lógicas inmediatas de otra(s) anterior(es). Llamamos *inferencia inmediata* a la obtención de una fórmula a partir de otra(s) por la aplicación de una sola regla de inferencia.

En el apartado 2 pasaremos a estudiar las reglas básicas de inferencia que vamos a utilizar, pero antes comentaremos brevemente el significado de los distintos tipos de líneas de nuestras deducciones.

1.2. Interpretación de las Líneas de Derivación

Para identificar las líneas de derivación utilizaremos las siguientes convenciones:

1.- *Numeración de líneas* : en el desarrollo de la derivación cada una de sus líneas irá numerada por la parte izquierda, en orden correlativo a partir del 1, de forma que el último número corresponda a la conclusión.

2 A (línea segunda de la derivación, consistente en la fórmula *A*)

2.- *Señalar premisas iniciales* : se pondrá en la parte izquierda de cada premisa una raya antes del nº de línea.

- 2 A (indica: supóngase como premisa la línea 2 de la derivación)

3.- *Comentarios a consecuencias inmediatas* : las líneas que se obtengan como consecuencia de aplicación de una regla, irán en su parte derecha seguidos del nombre de la regla que se aplica y sobre qué fórmulas (líneas) se aplica.

3 $A \rightarrow B$
... ...
8 A
9 B MP 3, 8 (por la regla «modus ponens» sobre las líneas 3 y 8, se introduce la nueva línea de derivación 9)

4.- *Señalar supuestos provisionales* : marcaremos la línea que incorporemos como supuesto mediante una escuadra mirando hacia abajo.

⎛ 8 A (supóngase, por el momento, que se verifica en la línea 8 la fórmula *A*)

5.- *Cancelar supuestos provisionales* : el precio pagado por hacer supuestos es que deben ser cancelados antes de finalizar la deducción. Una vez obtenida la conclusión subsidiaria que deseabamos se marca con una señal similar a la del inicio del supuesto pero invertida y a lo largo de la subprueba arrastramos el dibujo de la línea.

⎡ 8 A
⎢ ... ...
⎣ 15 $B \wedge \neg B$
16 $\neg A$

Ejemplos:

1.- Realizar la deducción formal del siguiente argumento

$$p \to (q \to r),\ p \to q,\ p \Rightarrow r$$

- 1	$p \to (q \to r)$	
- 2	$p \to q$	
- 3	p	
4	$q \to r$	MP 1, 3
5	q	MP 2, 3
6	r	MP 4, 5

2.- Caso particular de una derivación en la que no tenemos premisas iniciales

$$\Rightarrow p \to (q \to p \wedge q)$$

1	p	
2	q	
3	$p \wedge q$	IC 1, 2
4	$q \to p \wedge q$	TD 2-3
5	$p \to (q \to p \wedge q)$	TD 1-4

2. REGLAS BÁSICAS

Nos basaremos en el cálculo de Gentzen[1] que tiene 8 reglas llamadas *Reglas Básicas de Deducción Natural*, para el Cálculo de Proposiciones, dos para cada conectiva; y añadiremos 4 para el Cálculo de Predicados, dos para cada cuantificador. Si la regla básica introduce en su conclusión una conectiva o cuantificador que no aparece en sus premisas será una regla de introducción; si elimina de su conclusión una conectiva o cuantificador que aparece en sus premisas será una regla de eliminación. A partir de estas reglas básicas, podremos obtener otro tipo de reglas que se llaman *Reglas derivadas*, que nos servirán como "atajos" para acortar nuestras deducciones. Para realizar derivaciones o deducciones naturales podremos usar ambos tipos de reglas, básicas y derivadas, estas últimas únicamente una vez demostradas.

[1] Gerhard Gentzen, "Untersuchungen über das logische Schliessen" (Investigaciones sobre la deducción lógica) , Mathematische Zeitschrift, vol. 39, 1934, pág. 176-210.

2.1. REGLAS BÁSICAS DE IMPLICACIÓN

Regla de Eliminación del Implicador o «Modus Ponendo Ponens» (MP)

Forma:

$$\frac{A \rightarrow B \quad A}{B}$$

Significado: Supuesta cierta una implicación y supuesta también cierta la fórmula que hace en ella de antecedente, se puede afirmar el consecuente.

Ejemplo:

Premisa 1 Si Juan está en el partido de fútbol entonces Juan está en el estadio
Premisa 2 Juan está en el partido de fútbol
Conclusión Juan está en el estadio

Formalización:

p Juan está en el partido de fútbol
q Juan está en el estadio

$p \rightarrow q,\ p \Rightarrow q$

Derivación:

- 1	$p \rightarrow q$	
- 2	p	
3	q	MP 1, 2

Regla de Introducción del Implicador o «Teorema de Deducción» (TD)

Forma:

$$\frac{\begin{array}{|l} A \\ B \end{array}}{A \rightarrow B}$$

Significado: Si tengo una hipótesis cualquiera *A* y de ella se sigue *B*, puedo escribir como nueva fórmula $A \rightarrow B$. Esta regla constituye un ejemplo de suposición subsidiaria, la cual es la hipótesis de la que se parte, que es finalmente descargada o cancelada, cuando pasa a ser antecedente de una implicación. Es interesante para aquellos casos en los que la conclusión a la que se quiere llegar sea una implicación, porque así, se parte del antecedente y si mediante deducción se llega al consecuente, puede cancelarse la suposición y darse por concluida la implicación.

Ejemplo:

Resolver el siguiente argumento $p \wedge q \rightarrow r,\ r \rightarrow s \Rightarrow p \wedge q \rightarrow s$

Derivación:

- 1	$p \wedge q \rightarrow r$	
- 2	$r \rightarrow s$	
3	$p \wedge q$	
4	r	MP 1, 3
5	s	MP 2, 4
6	$p \wedge q \rightarrow s$	TD 3-5

2.2. REGLAS BÁSICAS DE CONJUNCIÓN

Regla de Introducción del Conjuntor (IC)

Forma:

$$\frac{\begin{array}{l}A\\B\end{array}}{A \wedge B}$$

Significado: Si en un contexto, se afirma primero una proposición y luego otra, se puede afirmar la conjunción de ambas.

Ejemplo:

Premisa 1	Jorge es adulto
Premisa 2	María es adolescente

Si ambas proposiciones son verdaderas, entonces se puede afirmar la proposición:

Jorge es adulto **y** María es adolescente

Regla de Eliminación del Conjuntor (EC)

Forma:

$$(EC_1)\ \frac{A \wedge B}{A} \qquad (EC_2)\ \frac{A \wedge B}{B}$$

Significado: Inversa a la anterior: si se dispone de la conjunción de dos proposiciones, se pueden afirmar cada una de ellas por separado.

Ejemplo:

Premisa	Jorge es adulto y María es adolescente
Se puede afirmar que	Jorge es adulto
y también que	María es adolescente

2.3. REGLAS BÁSICAS DE DISYUNCIÓN

Regla de Introducción del Disyuntor (ID)

Forma:

$$(ID_1)\ \frac{A}{A \vee B} \qquad (ID_2)\ \frac{B}{A \vee B}$$

Significado: Dada una fórmula cualquiera *A*, se puede pasar a una nueva fórmula por el sencillo procedimiento de adicionarle mediante el disyuntor la componente que nos plazca. Si una fórmula es verdadera, también es verdadera si le

añadimos con disyunción otra fórmula proposicional, tanto si es atómica como molecular.

Ejemplo:

Premisa Lirios es rubia

Se puede afirmar Lirios es rubia o morena

Regla de Eliminación del Disyuntor (ED)

Forma:

$$\begin{array}{l} A \vee B \\ \left[\begin{array}{l} A \\ C \end{array}\right. \\ \left[\begin{array}{l} B \\ C \end{array}\right. \\ \hline C \end{array}$$

Significado: Supuesta inicialmente una disyunción, en principio no se está autorizado a pasar a la afirmación de ninguna de sus componentes; lo que sí sabemos es que una al menos de sus componentes es verdadera, pero no sabemos cual. Para saberlo debemos de utilizar un recurso que es muy antiguo: se supone por separado que cada elemento de la disyunción es un supuesto provisional, si del análisis de estas suposiciones se llega al mismo resultado, se puede decir que ese resultado se sigue de la disyunción inicial (**Método por casos**).

Ejemplo:

Supongamos que al día siguiente por la mañana tenemos una entrevista con un cliente que llega de Madrid a Alicante. No sabemos el medio de locomoción que va a utilizar, pero nos enteramos que el avión llega por la mañana, y que si viene en tren también llega a media mañana; en cualquier caso, podemos tener la entrevista con él. Este razonamiento se apoya en el conocido método de la prueba por casos.

2.4. REGLAS BÁSICAS DE NEGACIÓN

Regla de Introducción del Negador (IN)

Forma:

$$\begin{array}{l} \left[\begin{array}{l} A \\ \ldots \\ B \wedge \neg B \end{array}\right. \\ \hline \neg A \end{array}$$

Significado: Se funda en la idea central de la lógica tradicional bivalente de que una contradicción es inadmisible; toda proposición que dé lugar a ella debe ser rechazada o negada. Se conoce como **Reducción al Absurdo**:

1.- Se supone la negación de la conclusión que se desea obtener.
2.- Se deduce a partir del supuesto una contradicción.
3.- Se niega el supuesto que ha dado lugar a la contradicción.
4.- Se establece, por tanto, la conclusión deseada.

Ejemplo:

Premisa 1	Si nieva se me enfrían las orejas
Premisa 2	No tengo las orejas frías

Puedo concluir que no nieva, ya que si nevara, se me enfriarían las orejas, y como al mismo tiempo sé que no tengo las orejas frías, llegaría a una contradicción.

Regla de Eliminación del Negador (EN)

Forma:
$$\frac{\neg \neg A}{A}$$

Significado: Negar doblemente algo es tanto como afirmarlo.

Ejemplo:

Premisa: No ocurre que Ana no es estudiante

La conclusión que se puede sacar de esta premisa es que:

Ana es estudiante

NOTA : el paso contrario, de la afirmación a doble negación, se estudiará como regla derivada.

2.5. REGLAS BÁSICAS DE CUANTIFICACIÓN UNIVERSAL

Regla de Eliminación del Universal (EU)

Forma:
$$\frac{\forall x \; P(x)}{P(a)}$$

Significado: De <u>todos en general</u> se pasa a <u>este individuo en particular</u>, es decir, si todos los elementos del universo de discurso verifican una propiedad *P*, desde luego que cualquiera de ellos, por ejemplo *a*, también la verifica.

Ejemplo: $\forall x (P(x) \rightarrow Q(x)),\ P(a) \Rightarrow Q(a)$

Derivación:

- 1	$\forall x (P(x) \rightarrow Q(x))$		
- 2	$P(a)$		
3	$P(a) \rightarrow Q(a)$	EU 1	
4	$Q(a)$	MP 2, 3	

Regla de Introducción del Universal (IU)

Forma: $\dfrac{P(a)}{\forall x\ P(x)}$ (con restricciones)

Significado: Si un individuo *a* verifica una propiedad *P*, entonces podemos afirmar que todos los *x* del universo la verifican si ese individuo que se ha elegido (*a*) es uno cualquiera y no uno en particular del universo, es decir, *a* es un individuo que tiene que estar libre de toda condición anterior.

Restricción: Podemos decir que es lícito pasar de *P(a)* a *∀x P(x)* siempre que el individuo *a* no figure en ningún supuesto provisional previo sin cancelar del que dependa *P(a)* o en una premisa (ya que las premisas se pueden considerar como supuestos pero que no necesitan ser cancelados).

Ejemplos:

1.- "O todos son moros o todos son cristianos.
Luego, todos son moros o cristianos."

Representación:

$C(x)$	*x* es cristiano
$M(x)$	*x* es moro

$\forall x\ C(x) \vee \forall x\ M(x) \Rightarrow \forall x [C(x) \vee M(x)]$

Derivación:

- 1	$\forall x\ C(x) \vee \forall x\ M(x)$		
2	$\forall x\ C(x)$		
3	$C(a)$	EU 2	
4	$C(a) \vee M(a)$	ID_1 3	
5	$\forall x\ M(x)$		
6	$M(a)$	EU 5	
7	$C(a) \vee M(a)$	ID_2 6	
8	$C(a) \vee M(a)$	Cas 1, 2-4, 5-7	
9	$\forall x (C(x) \vee M(x))$	IU 8	

NOTA : *a* estaba libre de supuestos provisionales sin cancelar.

2.- "Todos son moros o cristianos.
Luego, o todos son moros o todos son cristianos."

Representación: $\forall x (C(x) \vee M(x)) \Rightarrow \forall x C(x) \vee \forall x M(x)$

Derivación:			
- 1	$\forall x (C(x) \vee M(x))$		
2	$C(a) \vee M(a)$	EU 1	
3	$C(a)$		
4	$\forall x C(x)$	¡ ERROR !	

NOTA : No se puede introducir el universal puesto que el individuo al cual se le aplica, *a*, está sujeto a una condición sin cancelar en la línea 3.

2.6. REGLAS BÁSICAS DE CUANTIFICACIÓN EXISTENCIAL

Regla de Introducción del Existencial (IE)

Forma:

$$\frac{P(a)}{\exists x P(x)}$$

Significado: Si un individuo cualquiera verifica una propiedad *P*, desde luego que existe uno por lo menos que verifica dicha propiedad.

Ejemplo: $\forall x (P(x) \rightarrow Q(x)),\ P(a) \Rightarrow \exists x Q(x)$

Derivación:			
- 1	$\forall x (P(x) \rightarrow Q(x))$		
- 2	$P(a)$		
3	$P(a) \rightarrow Q(a)$	EU 1	
4	$Q(a)$	MP 2, 3	
5	$\exists x Q(x)$	IE 4	

Regla de Eliminación del Existencial (EE)

Forma:

$$\frac{\exists x P(x) \qquad \text{(con restricciones)}}{\begin{array}{|l} P(a) \\ \ldots \\ B \end{array}}$$

$$B$$

Significado: Si se sabe que hay algún sujeto que verifica una determinada propiedad entonces aún sin saber cual es exactamente ese sujeto, puedo pasar a las consecuencias que se siguen del supuesto de su identificación, es decir, del supuesto de que un sujeto, supuestamente imaginado, posea dicha propiedad.

Restricciones:

1.- El individuo elegido no puede ser cualquiera (pues alguno no es extensible a todos) sino uno tal que posea la propiedad en cuestión, y que ese sujeto no haya sido mencionado en otro supuesto sin cancelar o premisa, es decir, si *a* es ejemplo de *P* no puede ser ejemplo de otro tipo de condición.

2.- Para que la conclusión pueda ser aceptada, el individuo no debe aparecer en ella.

Ejemplos:

1.- $\exists x\, P(x) \wedge \exists x\, Q(x) \Rightarrow \exists x\, (P(x) \wedge Q(x))$

Derivación:

-	1	$\exists x\, P(x) \wedge \exists x\, Q(x)$	
	2	$\exists x\, P(x)$	EC_1 1
	3	$\exists x\, Q(x)$	EC_2 1
⎛	4	P(a)	
⎝	5	Q(a)	¡ ERROR !

NOTA: No puede coincidir la misma constante en *P* y en *Q*. Debe generarse una nueva constante cada vez que se realiza una especificación existencial.

2.- $\exists x\, (P(x) \rightarrow Q(x)),\ \forall x\, P(x) \Rightarrow \exists x\, Q(x)$

Derivación:

-	1	$\exists x\, (P(x) \rightarrow Q(x))$	
-	2	$\forall x\, P(x)$	
⎛	3	$P(a) \rightarrow Q(a)$	
⎝	4	P(a)	EU 2
	5	Q(a)	MP 3, 4
	6	Q(a)	¡ ERROR !

NOTA : La conclusión no puede depender de la elección que hayamos hecho para eliminar el existencial. Una posible derivación correcta sería:

-	1	$\exists x\, (P(x) \rightarrow Q(x))$	
-	2	$\forall x\, P(x)$	
⎛	3	$P(a) \rightarrow Q(a)$	
⎜	4	P(a)	EU 2
⎜	5	Q(a)	MP 3, 4
⎝	6	$\exists x\, Q(x)$	IE 5
	7	$\exists x\, Q(x)$	EE 1, 3-6

Tanto formalmente como intuitivamente, la regla de eliminación del existencial puede ser vista como una prueba por casos, ya que sabemos que algún objeto del dominio cumple la propiedad, pero no sabemos cuál de ellos. Por tanto intentamos demostrar que la conclusión se dará independientemente de cuál sea el objeto que la cumpla.

3. RESOLUCIÓN DE ARGUMENTOS

3.1. Subpruebas

Desde la introducción de un supuesto provisional hasta la línea en que lo cancelamos, lo podemos considerar como una *subdeducción*, *subprueba* o *subderivación*. Las subpruebas son unas características de los sistemas deductivos, que son muy utilies, pero que si no se usan correctamente pueden llevarnos a probar cosas que no se siguen de nuestras premisas.

Uno de los errores más comunes se produce al utilizar líneas interiores de subpruebas fuera de ellas. Una vez cancelado un supuesto, los pasos individuales internos (las líneas de derivación que aparecen entre los simbolos de supuesto y cancelación) ya no son accesibles. Únicamente se puede utilizar la línea que justifica la cancelación y que enumeraremos siempre a continuación de esta. En cambio, en una subprueba sí que podemos utilizar cualquier línea obtenida anteriormente en la deducción principal o en una subprueba todavía no cancelada.

Ejemplo:

$$(B \wedge A) \vee (A \wedge C) \Rightarrow A \wedge B$$

- 1	$(B \wedge A) \vee (A \wedge C)$		
┌ 2	$B \wedge A$		
│ 3	B	EC 2	
└ 4	A	EC 2	
┌ 5	$A \wedge C$		
└ 6	A	EC 5	
7	A	Casos 1, 2-4, 5-6	
8	$A \wedge B$	IC 7,3	¡¡¡ ERROR !!!

La línea 8 es errónea ya que no podemos utilizar la línea 3, ya que ésta ha desaparecido en el momento que hemos hecho la cancelación de la línea 4.

Los supuestos provisionales son una herramienta muy potente ya que nos permiten suponer lo que nosotros queramos. Pero debemos pagar un alto precio por ello: para poder finalizar una demostración deberemos haber cancelado todos los supuestos que hayamos hecho. Por tanto, la cancelación de supuestos provisionales se convierte en una pieza clave de las deducciones naturales. Veamoslo más detenidamente:

- Un supuesto provisional situado en la línea *m* de una derivación queda descargado o cancelado cuando en una línea posterior *n* de esa derivación, se obtiene una fórmula tal que permite la inferencia inmediata de otra buscada, que es independiente del referido supuesto y cuyo nº de línea será *n+1*. Esta última línea puede ser

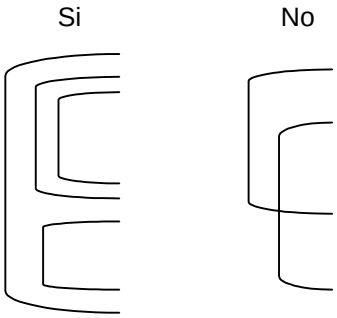
considerada como la conclusión de una deducción que comienza en la línea de derivación en que se introdujo el referido supuesto (deducción subsidiaria *m*-*n*).

- Esta derivación puede figurar sola o en el contexto de deducciones mayores.

- Las premisas de la deducción subsidiaria quedan afectadas por la cancelación del supuesto y no deben ser reutilizadas como antecedentes de nuevas inferencias, es decir, una vez cancelado un supuesto provisional, sus líneas interiores no son accesibles y por tanto no las podemos utilizar en nuestras posteriores inferencias.

- Una vez obtenida dicha conclusión subsidiaria se marca con una señal similar a la del supuesto pero invertida.

- Los supuestos provisionales no pueden solaparse, pero sí anidarse:



3.2. Resolución de Argumentos

El empleo de reglas para resolver argumentos mediante derivación puede sujetarse al siguiente plan:

1.- Asegurarse de que las fórmulas que componen el argumento están bien formalizadas, es decir, son fbf.

2.- Comprobar intuitivamente que el argumento está bien formulado, es decir, parece ser correcto:
   - si consideramos que parece ser correcto, abordaremos su demostración mediante deducción natural.
   - si tiene visos de ser incorrecto, intentaremos encontrar un contraejemplo.

3.- Enumerar las premisas iniciales y señalarlas.

4.- Extraer de ellas la conclusión deseada. Para ello puede que sea necesario la sucesiva aplicación inmediata de reglas de inferencia. Un orden aconsejable para la aplicación de las reglas puede ser:
   - Aplicar reglas de eliminación de cuantificadores.
   - Aplicar las reglas que se crean oportunas a las fórmulas ya sin cuantificación
   - Introducir cuantificadores hasta llegar a la conclusión buscada

4. REGLAS DERIVADAS

Las doce reglas que se han visto para la deducción natural son por sí solas suficientes para resolver cualquier problema de deducción formal que se presente dentro del cálculo de predicados. Sin embargo, solamente con estas reglas, es muy lenta la resolución de argumentos, por ello lo que hacemos es crear combinaciones de reglas básicas y así obtener lo que se llaman *reglas derivadas*. Una vez demostrada una regla derivada ya podemos utilizarla en nuestras deducciones. Con estas reglas vamos a conseguir algo muy importante que es el reducir el número de líneas de una derivación. Podemos ver a estas reglas derivadas como "atajos" que tomamos para llegar al destino, de forma que lo único que aportan son rapidez para alcanzar el destino. No hay nada que podamos demostrar con reglas derivadas que no podamos demostrar utilizando reglas básicas únicamente.

Ejemplo:

Supongamos que tenemos que es cierto $A \rightarrow B$ y también que es cierto $\neg B$; con estas dos premisas queremos determinar si la conclusión $\neg A$ es consecuencia de ellas, para lo cual hacemos una deducción formal:

-	1	$A \rightarrow B$	
-	2	$\neg B$	
┌	3	A	
|	4	B	MP 1, 3
└	5	$B \wedge \neg B$	IC 2, 4
	6	$\neg A$	Abs. 3-5

Este proceso es válido para cualquier par de fórmulas A y B, luego siempre que nos encontremos en una situación igual, podemos ahorrarnos esta rutina con solo añadir al grupo de reglas básicas ésta que acabamos de demostrar.

Esta regla vista y demostrada en el ejemplo, a partir de este momento ya puede ser considerada como una figura deductiva, que no será básica sino derivada, o sea fundada en la aplicación de reglas básicas.

A continuación se expondrá la fundamentación de algunas reglas derivadas; otras se darán en la tabla adjunta, dejando la fundamentación como ejercicio para el

lector. A la fundamentación de las reglas derivadas en otras anteriores, se le puede llamar también deducción, aunque sea más bien esquema de deducciones.

Las reglas derivadas del cálculo de proposiciones (o del cálculo de predicados sin cuantificación) se agruparán en reglas de implicación, reglas de conjunción y disyunción y reglas de negación. También existen otras reglas adicionales con combinaciones de distintos juntores y de estructura más o menos compleja: reglas de coimplicación, regla de intercambio, leyes de interdefinición, leyes de De Morgan. Por último veremos algunas reglas en las que intervienen cuantificadores: reglas de interdefinición de cuantificadores y reglas de cuantificación múltiple.

La doble raya horizontal (o el símbolo $\Leftrightarrow$) significará que la regla también es válida en sentido inverso, es decir, de abajo hacia arriba (de derecha a izquierda).

4.1. REGLAS DE IMPLICACIÓN

Silogismo Hipotético (Sil)

$A \rightarrow B$
$\underline{B \rightarrow C}$
$A \rightarrow C$

Fundamentación:

-	1	$A \rightarrow B$	
-	2	$B \rightarrow C$	
⌐	3	A	
|	4	B	MP 1, 3
⌙	5	C	MP 2, 4
	6	$A \rightarrow C$	TD 3-5

Identidad (Id)

$\underline{A}$
A

Mutación de Premisas (Mut)

$\underline{A \rightarrow (B \rightarrow C)}$
$B \rightarrow (A \rightarrow C)$

Carga de Premisa (CPr)

$\underline{A \quad\quad}$
$B \rightarrow A$

Fundamentación:

-	1	A	
⌐	2	B	
⌙	3	A	Id 1
	4	$B \rightarrow A$	TD 2-3

4.2. REGLAS DE CONJUNCIÓN Y DISYUNCIÓN

Conmutativa de la Conjunción (CC) y de la Disyunción (CD)

$A \wedge B \Leftrightarrow B \wedge A$ $\qquad$ $A \vee B \Leftrightarrow B \vee A$

Asociativa de la Conjunción (AC) y de la Disyunción (AD)

$(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$ $\qquad$ $(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$

Distributiva de la Conjunción en Disyunción (DC)

$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Distributiva de la Disyunción en Conjunción (DD)

$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

Idempotencia de la Conjunción (IdC) y de la Disyunción (IdD)

$A \wedge A \Leftrightarrow A$ $\qquad$ $A \vee A \Leftrightarrow A$

Absorción de la Conjunción (AbsC) y de la Disyunción (AbsD)

$A \wedge (A \vee B) \Leftrightarrow A$ $\qquad$ $A \vee (A \wedge B) \Leftrightarrow A$

A continuación se pasará a estudiar la fundamentación de algunas de estas reglas. Como son reglas que se verifican en ambos sentidos, la demostración tiene que ser también en ambos sentidos.

Conmutativa de la disyunción:

$$\frac{A \vee B}{B \vee A}$$

Fundamentación de $A \vee B \Rightarrow B \vee A$

- 1	$A \vee B$	
2	A	
3	$B \vee A$	ID_2 2
4	B	
5	$B \vee A$	ID_1 4
6	$B \vee A$	Cas 1, 2-3, 4-5

Fundamentación de $B \vee A \Rightarrow A \vee B$ (de manera similar)

Conmutativa de la conjunción:

$A \wedge B$

$B \wedge A$

Fundamentación de $A \wedge B \Rightarrow B \wedge A$

- 1	$A \wedge B$		
2	A		EC_1 1
3	B		EC_2 1
4	$B \wedge A$		IC 2, 3

Fundamentación de $B \wedge A \Rightarrow A \wedge B$ (de manera similar)

Asociativa de la disyunción:

$(A \vee B) \vee C$

$A \vee (B \vee C)$

Fundamentación de $(A \vee B) \vee C \Rightarrow A \vee (B \vee C)$

- 1	$(A \vee B) \vee C$		
2	$A \vee B$		
3	A		
4	$A \vee (B \vee C)$		ID_1 3
5	B		
6	$B \vee C$		ID_1 5
7	$A \vee (B \vee C)$		ID_2 6
8	$A \vee (B \vee C)$		Cas 2, 3-4, 5-7
9	C		
10	$B \vee C$		ID_2 9
11	$A \vee (B \vee C)$		ID_2 10
12	$A \vee (B \vee C)$		Cas 1, 2-8, 9-11

Fundamentación de $A \vee (B \vee C) \Rightarrow (A \vee B) \vee C$ (similar)

Idempotencia de la conjunción:

$A \wedge A$

A

Fundamentación de $A \wedge A \Rightarrow A$

- 1	$A \wedge A$		
2	A		EC_1 1

Fundamentación de $A \Rightarrow A \wedge A$

- 1	A		
2	A		Id 1
3	$A \wedge A$		IC 1, 2

4.3. REGLAS DE NEGACIÓN

Contraposición (Cp)

$$\frac{A \rightarrow B}{\neg B \rightarrow \neg A}$$

Fundamentación:

- 1	$A \rightarrow B$	
2	$\neg B$	
3	A	
4	B	MP 1, 3
5	$B \wedge \neg B$	IC 4, 2
6	$\neg A$	Abs 3-5
7	$\neg B \rightarrow \neg A$	TD 2-6

Modus Tollens (MT)

$$\frac{A \rightarrow B \qquad \neg B}{\neg A}$$

Fundamentación:

- 1	$A \rightarrow B$	
- 2	$\neg B$	
3	A	
4	B	MP 1, 3
5	$B \rightarrow \neg B$	IC 4, 2
6	$\neg A$	Abs 3-5

Introducción de Doble Negador (IDN)

$$A \Rightarrow \neg \neg A$$

Ex Contradictione Quodlibet (ECQ) (Eliminación débil del negador)

$$A \wedge \neg A \Rightarrow B$$

Principio de No Contradicción (PNC)

$$\neg (A \wedge \neg A)$$

Principio de Exclusión de tercero (PTE)

$$A \vee \neg A$$

NOTA : Ver las reglas derivadas adicionales en la hoja adjunta al final de este capítulo.

4.4. REGLAS DE CUANTIFICACIÓN

Interdefinición del Existencial (DE)

Forma:

$$\frac{\exists x\ P(x)}{\neg\forall x\ \neg P(x)}$$

Demostración:

1º.- $\exists x\ P(x) \Rightarrow \neg\forall x\ \neg P(x)$

-	1	$\exists x\ P(x)$	
┌	2	$P(a)$	
│ ┌	3	$\forall x\ \neg P(x)$	
│ │	4	$\neg P(a)$	EU 3
│ └	5	$P(a) \wedge \neg P(a)$	IC 2, 4
└	6	$\neg\forall x\ \neg P(x)$	Abs 3-5
	7	$\neg\forall x\ \neg P(x)$	EE 1, 2-6

2º.- $\neg\forall x\ \neg P(x) \Rightarrow \exists x\ P(x)$

-	1	$\neg\forall x\ \neg P(x)$	
┌	2	$\neg\exists x\ P(x)$	
│ ┌	3	$P(a)$	
│ │	4	$\exists x\ P(x)$	IE 3
│ └	5	$\exists x\ P(x) \wedge \neg\exists x\ P(x)$	IC 4, 2
│	6	$\neg P(a)$	Abs 3-5
│	7	$\forall x\ \neg P(x)$	IU 6
└	8	$\forall x\ P(x) \wedge \neg\forall x\ P(x)$	IC 1, 7
	9	$\neg\neg\exists x\ P(x)$	Abs 2-8
	10	$\exists x\ P(x)$	EN 9

Interdefinición del Universal (DU)

Forma:

$$\frac{\forall x\ P(x)}{\neg\exists x\ \neg P(x)}$$

Negador del Universal (NU)

Forma:

$$\frac{\neg\forall x\ P(x)}{\exists x\ \neg P(x)}$$

Negador del Existencial (NE)

Forma: $\dfrac{\neg\exists x\ P(x)}{\forall x\ \neg P(x)}$

Las reglas anteriores las podemos considerar como una generalización de las Leyes de De Morgan para cuantificadores. Cuando el universo es discreto y finito la fórmula *∀x P(x)* es equivalente a una conjunción del predicado *P* para cada uno de los elementos del universo (y de manera similar *∃x P(x)* como una disyunción), por ello es sencillo generalizar las leyes de De Morgan para la negación de universales o de existenciales.

Cuantificación Múltiple (CM)

CM1: $\forall x\ \forall y\ P(x, y) \Leftrightarrow \forall y\ \forall x\ P(x, y)$

CM2: $\exists x\ \exists y\ P(x, y) \Leftrightarrow \exists y\ \exists x\ P(x, y)$

CM3: $\exists x\ \forall y\ P(x, y) \Rightarrow \forall y\ \exists x\ P(x, y)$

CM4: $\forall x\ \forall y\ P(x, y) \Rightarrow \forall x\ P(x, x)$

CM5: $\exists x\ P(x, x) \Rightarrow \exists x\ \exists y\ P(x, y)$

Demostración CM2:

1º $\exists x\ \exists y\ P(x, y) \Rightarrow \exists y\ \exists x\ P(x, y)$

- 1	$\exists x\ \exists y\ P(x, y)$	
2	$\exists y\ P(a, y)$	
3	$P(a, b)$	
4	$\exists x\ P(x, b)$	IE 3
5	$\exists y\ \exists x\ P(x, y)$	IE 4
6	$\exists y\ \exists x\ P(x, y)$	EE 2, 3-5
7	$\exists y\ \exists x\ P(x, y)$	EE 1, 2-6

2º Para el otro sentido se hace igual.

Demostración CM3: $\exists x\ \forall y\ P(x, y) \Rightarrow \forall y\ \exists x\ P(x, y)$

- 1	$\exists x\ \forall y\ P(x, y)$	
2	$\forall y\ P(a, y)$	
3	$P(a, b)$	EU 2
4	$\exists x\ P(x, b)$	IE 3
5	$\forall y\ \exists x\ P(x, y)$	IU 4
6	$\forall y\ \exists x\ P(x, y)$	EE 1, 2-5

5. ESTRATEGIAS BÁSICAS DE DEMOSTRACIÓN

Cuando queramos demostrar que un argumento es válido, podemos atacar el problema utilizando distintas estrategias. La elección dependerá de los gustos de cada persona, y de la forma que tengan las premisas y la conclusión. Antes de describir las distintas técnicas recordemos que un argumento es válido tanto si las premisas y la conclusión son verdaderas como si alguna premisa es falsa y la conclusión es verdadera o falsa. Y no es correcto cuando las premisas son verdaderas y la conclusión falsa. Basándonos en esta definición y en algunas reglas básicas ya vistas, disponemos de los siguientes métodos para realizar demostraciones de argumentos:

5.1. Método 1: PRUEBA TRIVIAL

Fundamentación: Para que $X \Rightarrow Y$ sea válido, basta con que Y sea tautología, es decir, si es posible obtener el valor de verdad de Y, y este es verdadero, entonces independientemente del valor de X, el argumento será correcto.

Ejemplo : Demostrar utilizando el Método Trivial si el siguiente argumento es válido:

$$[\, A \vee (B \wedge \neg C) \longrightarrow (A \vee \neg B)\,] \;\Rightarrow\; (A \wedge B) \vee \neg A \vee \neg B$$

Bastará comprobar que el valor de verdad del consecuente, Y = $(A \wedge B) \vee \neg A \vee \neg B$, es verdadero:

$$(A \wedge B) \vee \neg A \vee \neg B$$

$$(A \wedge B) \vee \neg(A \wedge B)$$

Verdadero

Luego, como hemos visto que el valor de verdad de Y es verdadero, entonces podemos decir que el argumento es válido.

5.2. Método 2: PRUEBA VACÍA

Fundamentación: Para demostrar que $X \Rightarrow Y$ es correcto basta demostrar que X es una contradicción.

Ejemplo: Demostrar utilizando la Prueba Vacía si el siguiente argumento es correcto:

$$A \wedge B \wedge (\neg A \vee \neg B) \;\Rightarrow [\, A \vee (B \wedge \neg C) \longrightarrow (A \vee \neg B)\,]$$

Bastará comprobar que el valor de verdad del antecedente, X = $A \wedge B \wedge (\neg A \vee \neg B)$, es falso:

$$A \wedge B \wedge (\neg A \vee \neg B)$$

$$(A \wedge B) \wedge \neg(A \wedge B)$$

Falso

Luego, como hemos visto que el valor de verdad de X es falso, entonces podemos afirmar que si se da X se dará Y.

5.3. Método 3: PRUEBA DIRECTA

Si la conclusión tiene forma de implicación, puede ponerse como supuesto provisional el antecedente de dicha fórmula, con lo cual se reduce el problema a obtener el consecuente de dicha implicación y establecer, por la regla de introducción del implicador ("Teorema de Deducción"), la fórmula deseada, al tiempo que se cancela el supuesto.

Fundamentación: Sea la conclusión $X \rightarrow Y$. Empezar por suponer que X es V y entonces, a partir de inferencias válidas, obtener que la conclusión es V. Cancelamos por TD y obtenemos la fórmula deseada.

Ejemplo:

$$(P \wedge Q \longrightarrow \neg R) \wedge (P \vee Q \longrightarrow R) \longrightarrow (P \longrightarrow \neg Q)$$

Derivación:

1	$(P \wedge Q \rightarrow \neg R) \wedge (P \vee Q \rightarrow R)$	
2	$P \wedge Q \rightarrow \neg R$	EC_1 1
3	$P \vee Q \rightarrow R$	EC_2 1
4	P	
5	$P \vee Q$	ID_1 4
6	R	MP 3,5
7	$\neg(P \wedge Q)$	MT 2,6
8	$\neg P \vee \neg Q$	DM 7
9	$\neg Q$	SD_2 4,8
10	$P \rightarrow \neg Q$	TD 4-9
11	$(P \wedge Q \rightarrow \neg R) \wedge (P \vee Q \rightarrow R) \rightarrow (P \rightarrow \neg Q)$	TD 1-10

5.4. Método 4: PRUEBA INDIRECTA

Otra técnica que podemos utilizar cuando la conclusión tiene forma de implicación es que como en la definición de implicación lo que no podemos admitir es que el antecedente sea verdadero y el consecuente falso, de ser falso el consecuente, también deben ser falsas las premisas.

Fundamentación: Prueba directa del contrapositivo: como la implicación $X \longrightarrow Y$ es equivalente a $\neg Y \longrightarrow \neg X$, para establecer una prueba indirecta, basta obtener que $\neg Y \longrightarrow \neg X$ por el método directo.

Ejemplo:

Para probar que $(A \longrightarrow \neg A) \rightarrow \neg A$ es válida, basta con que probar que $\neg\neg A \rightarrow \neg(A \longrightarrow \neg A)$ también lo es.

Derivación:

1	$\neg\neg A$	
2	$A \rightarrow \neg A$	
3	A	EN 1
4	$\neg A$	MP 2,3
5	$A \wedge \neg A$	IC 3,4
6	$\neg(A \rightarrow \neg A)$	IN 2-5
7	$\neg\neg A \rightarrow \neg(A \rightarrow \neg A)$	TD 1-6

5.5. Método 5: PRUEBA POR CONTRADICCIÓN

Fundamentación: Se supone que la conclusión a la que queremos llegar es falsa y hay que conseguir una contradicción (con la ayuda de las premisas); por tanto si las premisas y la negación de la conclusión son contradictorias, será cierto que de las premisas se puede llegar a la conclusión. Este método también se conoce por **Método de Reducción al Absurdo**.

Ejemplo:

$$[[(P \wedge Q) \longrightarrow \neg R] \wedge [(P \vee Q) \longrightarrow R]] \Rightarrow (P \longrightarrow \neg Q)$$

Derivación:

- 1	$[(P \wedge Q) \rightarrow \neg R] \wedge [(P \vee Q) \rightarrow R]$	
2	$P \wedge Q \rightarrow \neg R$	EC_1 1
3	$P \vee Q \rightarrow R$	EC_2 1
4	$\neg(P \rightarrow \neg Q)$	
5	$\neg(\neg P \vee \neg Q)$	I, DI 4
6	$\neg\neg P \wedge \neg\neg Q$	DM 5
7	$P \wedge Q$	I^2 EN 6
8	P	EC_1 7
9	$P \vee Q$	ID_1 8
10	R	MP 3,9
11	$\neg R$	MP 2,7
12	$R \wedge \neg R$	IC 10,11
13	$\neg\neg(P \rightarrow \neg Q)$	Abs. 4-12
14	$P \rightarrow \neg Q$	EN 13

5.6. Método 6: PRUEBA POR CASOS

Fundamentación: Si en las premisas a utilizar se da una disyunción, $X=X_1 \vee X_2 \vee ... \vee X_n$, la verdad de la conclusión Y la podemos demostrar estableciendo como supuestos provisionales cada uno de los términos de la disyunción y procurando llegar a la conclusión deseada por todos y cada uno de los caminos.

Ejemplo:

$$(P \longrightarrow Q) \Rightarrow [\, P \longrightarrow (P \wedge Q)\,]$$

Derivación:

-	1	$P \rightarrow Q$	
	2	$\neg P \vee Q$	DI 1
	3	$\neg P$	
	4	$\neg P \vee (P \wedge Q)$	ID 3
	5	$P \rightarrow (P \wedge Q)$	DI 4
	6	Q	
	7	P	
	8	$P \wedge Q$	IC 6, 7
	9	$P \rightarrow (P \wedge Q)$	TD 7-8
	10	$P \rightarrow (P \wedge Q)$	Casos 2, 3-5, 6-9

Como hemos enumerado anteriormente, un paso previo al realizar una deducción formal es ver intuitivamente que realmente la conclusión se sigue de las premisas. Para ello nos podemos ayudar de las distintas estrategias o técnicas que acabamos de ver. Una vez diseñado el plan a seguir ya estamos en condiciones de trasladar este razonamiento informal y obtener la prueba formal que buscabamos. En ningún momento debemos perder de vista de donde hemos partido (*premisas*) y a donde queremos llegar (*conclusión*), ya que esta información guiará nuestros pasos ayudandonos a trazar el camino adecuado.

Para finalizar este apartado, comentar que estas estrategias básicas casi nunca se aplican por separado sino que sirven para ir acercandonos a la solución final, de forma que una deducción podrá constar de varias técnicas combinadas.

Ejemplo:

El afamado inspector Clouseau fue solicitado para analizar los siguientes hechos relacionados con un robo que tenía como sospechosos a tres conocidos “chorizos”: Makinavaja, Popeye y El Pirata.

1- Si Maki es culpable y Popi inocente, entonces El Pira es culpable.
2- El Pira nunca trabaja solo.
3- Maki no trabaja con El Pirata.
4- Se sabe que al menos uno de estos tres personajes es el culpable, y que nadie distinto a ellos intervino.

Evidentemente esta información es insuficiente para determinar quienes son culpables y quienes son inocentes. Pero, ¿ puedes acusar a alguno de ellos ?

Solución:

Veamos primero una aproximación informal a la solución:

Sí, **Popeye es culpable**. Supongamos que Popeye es inocente, entonces si Maki fuese culpable, por 1 El Pirata debería ser también culpable, pero esto se contradice con 3 ya que Maki y El Pirata no trabajan nunca juntos; por tanto Makinavaja también es inocente. Por 4 El Pirata debe

ser culpable ya que tanto Popeye como Maki son inocentes; pero por 2 como El Pirata nunca trabaja solo y por 3 no trabaja con Maki, deberá ser Popeye culpable.

Pasemos ahora a la formalización:

Sean las variables proposicionales:

p	Makinavaja es culpable
q	Popeye es culpable
r	El Pirata es culpable

-	1	$p \wedge \neg q \rightarrow r$	
-	2	$r \rightarrow p \vee q$	
-	3	$\neg(p \wedge r)$	
-	4	$p \vee q \vee r$	
	5	$\neg p \vee \neg r$	DM 3
┌	6	$p \wedge \neg q$	
│	7	r	MP 1, 6
│	8	$\neg p$	SD 5, 7
│	9	p	EC 6
└	10	$p \wedge \neg p$	IC 8, 9
	11	$\neg(p \wedge \neg q)$	Abs 6-10
	12	$\neg p \vee \neg\neg q$	DM 11
	13	$\neg p \vee q$	I, EN 12
┌	14	$\neg p$	
│	15	$q \vee r$	SD 4, 14
│ ┌	16	q	
│ └	17	q	Id 16
│ ┌	18	r	
│ │	19	$p \vee q$	MP 2, 18
│ └	20	q	SD 14, 19
└	21	q	Casos 15, 16-17, 18-20
┌	22	q	
└	23	q	Id 22
	24	q	Casos 13, 14-21, 22-23

6. EJERCICIOS

Comprobar, por Deducción Natural, que los siguientes argumentos son correctos.

1.- Aplicación del Modus Ponens (MP)

1.1.- $\mathbf{p \vee q,\ r \rightarrow s \wedge t,\ p \vee q \rightarrow r \Rightarrow s \wedge t}$

Solución:

-	1	$p \vee q$	
-	2	$r \rightarrow s \wedge t$	
-	3	$p \vee q \rightarrow r$	
	4	r	MP 1, 3
	5	$s \wedge t$	MP 2, 4 (Conclusión pedida)

1.2.- $\mathbf{p \to q,\ q \to r,\ p \Rightarrow r}$

Solución:

- 1	$p \to q$	
- 2	$q \to r$	
- 3	p	
4	q	MP 1, 3
5	r	MP 2, 4

2.- Aplicación de la Introducción del Implicador (Teorema de Deducción)

2.1.- $\mathbf{p \to q,\ q \to r \wedge t,\ r \wedge t \to s,\ s \to t \Rightarrow p \to t}$

Solución:

- 1	$p \to q$	
- 2	$q \to r \wedge t$	
- 3	$r \wedge t \to s$	
- 4	$s \to t$	
5	p	
6	q	MP 1, 5
7	$r \wedge t$	MP 2, 6
8	s	MP 3, 7
9	t	MP 4, 8
10	$p \to t$	TD 5-9

2.2.- $\mathbf{p \to (q \to (r \to s)) \Rightarrow q \to (r \to (p \to s))}$

Solución:

- 1	$p \to (q \to (r \to s))$	
2	q	
3	r	
4	p	
5	$q \to (r \to s)$	MP 1, 4
6	$r \to s$	MP 2, 5
7	s	MP 3, 6
8	$p \to s$	TD 4-7
9	$r \to (p \to s)$	TD 3-8
10	$q \to (r \to (p \to s))$	TD 2-9

3.- Aplicación de Eliminación de la Conjunción (EC)

$\mathbf{p \to q \wedge r,\ r \to s \wedge t \Rightarrow p \to t}$

Solución:

- 1	$p \rightarrow q \wedge r$	
- 2	$r \rightarrow s \wedge t$	
3	p	
4	$q \wedge r$	MP 1, 3
5	r	EC_2 4
6	$s \wedge t$	MP 2, 5
7	t	EC_2 6
8	$p \rightarrow t$	TD 3-7

4.- Aplicación de Introducción de la Conjunción (IC)

$$\mathbf{p \rightarrow q \wedge r,\ q \wedge r \rightarrow s \vee t,\ s \vee t \rightarrow w,\ p \Rightarrow p \wedge w}$$

Solución:

- 1	$p \rightarrow q \wedge r$	
- 2	$q \wedge r \rightarrow s \vee t$	
- 3	$s \vee t \rightarrow w$	
- 4	p	
5	$q \wedge r$	MP 1, 4
6	$s \vee t$	MP 2, 5
7	w	MP 3, 6
8	$p \wedge w$	IC 4, 7

5.- Aplicación de Eliminación e Introducción de la Conjunción

$$\mathbf{p \rightarrow (q \rightarrow r) \wedge (r \rightarrow \neg s),\ (\neg r \vee s) \wedge (p \wedge (\neg q \rightarrow t)) \Rightarrow (\neg q \rightarrow t) \wedge (r \rightarrow \neg s)}$$

Solución:

- 1	$p \rightarrow (q \rightarrow r) \wedge (r \rightarrow \neg s)$	
- 2	$(\neg r \vee s) \wedge (p \wedge (\neg q \rightarrow t))$	
3	$p \wedge (\neg q \rightarrow t)$	EC_2 2
4	p	EC_1 3
5	$(q \rightarrow r) \wedge (r \rightarrow \neg s)$	MP 1, 4
6	$\neg q \rightarrow t$	EC_1 3
7	$r \rightarrow \neg s$	EC_2 5
8	$(\neg q \rightarrow t) \wedge (r \rightarrow \neg s)$	IC 6, 7

6.- Aplicación de Introducción del Disyuntor (ID)

$$\mathbf{p \wedge q \rightarrow r \wedge s,\ q \Rightarrow p \rightarrow s \vee t}$$

Solución:

-	1	$p \wedge q \rightarrow r \wedge s$	
-	2	q	
	3	p	
	4	$p \wedge q$	IC 2, 3
	5	$r \wedge s$	MP 1, 4
	6	s	EC_2 5
	7	$s \vee t$	ID_1 6
	8	$p \rightarrow s \vee t$	TD 3-7

7.- Aplicación de Eliminación del Disyuntor (Casos)

$$\mathbf{p \vee \neg q \rightarrow (q \vee r) \wedge (\neg s \vee \neg t),\ q \rightarrow s,\ r \rightarrow s \ \Rightarrow\ p \rightarrow s}$$

Solución:

-	1	$p \vee \neg q \rightarrow (q \vee r) \wedge (\neg s \vee \neg t)$	
-	2	$q \rightarrow s$	
-	3	$r \rightarrow s$	
	4	p	
	5	$p \vee \neg q$	ID_1 4
	6	$(q \vee r) \wedge (\neg s \vee \neg t)$	MP 1, 5
	7	$q \vee r$	EC_1 1, 6
	8	q	
	9	s	MP 2, 8
	10	r	
	11	s	MP 3, 10
	12	s	Cas 7, 8-9, 10-11
	13	$p \rightarrow s$	TD 4-12

8.- Aplicación de Introducción del Negador (Reducción al absurdo) y Eliminación del Negador (EN). Utilizando únicamente reglas básicas demostrar:

$$\mathbf{\neg(p \vee \neg q) \ \Rightarrow\ \neg p \wedge q}$$

Solución:

-	1	$\neg(p \vee \neg q)$	
	2	p	
	3	$p \vee \neg q$	ID_1 2
	4	$\neg(p \vee \neg q) \ \wedge\ (p \vee \neg q)$	IC 1, 3
	5	$\neg p$	Abs. 2-4
	6	$\neg q$	
	7	$p \vee \neg q$	ID_2 6
	8	$\neg(p \vee \neg q) \ \wedge\ (p \vee \neg q)$	IC 1, 7
	9	$\neg\neg q$	Abs. 6-8
	10	q	EN 9
	11	$\neg p \wedge q$	IC 5, 10

9.- Realizar la derivación formal del argumento $\mathbf{p \rightarrow (q \rightarrow \neg r),\ q \wedge r \Rightarrow \neg p}$

a) Utilizando solamente reglas básicas

Solución:

- 1	$p \rightarrow (q \rightarrow \neg r)$	
- 2	$q \wedge r$	
3	p	
4	$q \rightarrow \neg r$	MP 1, 3
5	q	EC_1 2
6	$\neg r$	MP 4, 5
7	r	EC_2 2
8	$r \wedge \neg r$	IC 7, 6
9	$\neg p$	Abs 3-8

b) Con reglas derivadas

Solución:

- 1	$p \rightarrow (q \rightarrow \neg r)$	
- 2	$q \wedge r$	
3	$q \rightarrow (p \rightarrow \neg r)$	Mut 1
4	q	EC_1 2
5	$p \rightarrow \neg r$	MP 3, 4
6	r	EC_2 2
7	$\neg\neg r$	IDN 6
8	$\neg p$	MT 5, 7

10.- Como veremos en capítulos posteriores, una de las reglas de inferencia más importante es la «resolución». Vamos a demostrarla:

$$\mathbf{L \vee A,\ \neg L \vee B \Rightarrow A \vee B}$$

Solución:

- 1	$L \vee A$	
- 2	$\neg L \vee B$	
3	$\neg L$	
4	A	SD 1, 3
5	$A \vee B$	ID_1 4
6	B	
7	$A \vee B$	ID_2 6
8	$A \vee B$	Casos 2, 3-5, 6-7

11.- Resolver por deducción natural el siguiente argumento:

$$\mathbf{\forall x\,(\,R(x) \rightarrow P(x)\,),}$$
$$\mathbf{\forall x\,(\,P(x) \rightarrow \neg\, S(x)\,),}$$
$$\mathbf{\forall x\,(\,R(x) \wedge Q(x) \rightarrow S(x)\,)} \quad \Rightarrow \quad \mathbf{\forall x\,(\ R(x) \rightarrow \neg(\,P(x) \rightarrow Q(x)\,)\ )}$$

Solución:

-	1	$\forall x (R(x) \rightarrow P(x))$	
-	2	$\forall x (P(x) \rightarrow \neg S(x))$	
-	3	$\forall x (R(x) \wedge Q(x) \rightarrow S(x))$	
	4	$R(a) \rightarrow P(a)$	EU 1
	5	$P(a) \rightarrow \neg S(a)$	EU 2
	6	$R(a) \wedge Q(a) \rightarrow S(a)$	EU 3
	7	$R(a)$	
	8	$P(a)$	MP 4, 7
	9	$\neg S(a)$	MP 5, 8
	10	$\neg(R(a) \wedge Q(a))$	MT 6, 9
	11	$\neg R(a) \vee \neg Q(a)$	DM_1 10
	12	$R(a) \rightarrow \neg Q(a)$	DI_2 11
	13	$\neg Q(a)$	MP 7, 12
	14	$P(a) \wedge \neg Q(a)$	IC 8, 13
	15	$\neg(P(a) \rightarrow Q(a))$	DfC_1 14
	16	$R(a) \rightarrow \neg(P(a) \rightarrow Q(a))$	TD 7-15
	17	$\forall x (R(x) \rightarrow \neg(P(x) \rightarrow Q(x)))$	IU 16

12.- Resolver el siguiente argumento por deducción natural:

$$\neg\exists x (P(x) \wedge \neg R(x)) \rightarrow \neg\forall x (Q(x) \rightarrow P(x)),\ \forall x (P(x) \rightarrow R(x)) \Rightarrow \exists x \neg(\neg Q(x) \vee P(x))$$

Solución:

-	1	$\neg\exists x (P(x) \wedge \neg R(x)) \rightarrow \neg\forall x (Q(x) \rightarrow P(x))$	
-	2	$\forall x (P(x) \rightarrow R(x))$	
	3	$\neg \neg\exists x (P(x) \wedge \neg R(x)) \vee \neg\forall x (Q(x) \rightarrow P(x))$	DI_2 1
	4	$\exists x (P(x) \wedge \neg R(x)) \vee \neg\forall x (Q(x) \rightarrow P(x))$	I, EN 3
	5	$\exists x (P(x) \wedge \neg R(x))$	
	6	$P(a) \wedge \neg R(a)$	
	7	$P(a) \rightarrow R(a)$	EU 2
	8	$P(a)$	EC 6
	9	$R(a)$	MP 7, 8
	10	$\neg R(a)$	EC 6
	11	$R(a) \wedge \neg R(a)$	IC 9, 10
	12	$\exists x \neg(\neg Q(x) \vee P(x))$	ECQ 11
	13	$\exists x \neg(\neg Q(x) \vee P(x))$	EE 5, 6-12
	14	$\neg\forall x (Q(x) \rightarrow P(x))$	
	15	$\exists x \neg(Q(x) \rightarrow P(x))$	NU 14
	16	$\exists x \neg(\neg Q(x) \vee P(x))$	I, DI_2 15
	17	$\exists x \neg(\neg Q(x) \vee P(x))$	Cas 4,5-13,14-16

13.- Aquí tenemos tres silogismos que aparecen en el libro “Symbolic Logic” de Lewis Carroll. Hay que averiguar si la conclusión, en cada caso, es o no correcta. Si es correcta demostrarlo por Deducción Natural; si no lo es buscar un contraejemplo.

a) **Algunas almohadas son blandas.**

Ningún atizador es blando.
***Luego,* algunos atizadores no son almohadas.**

Solución:

Al(x)	*x* es almohada
B(x)	*x* es blando
At(x)	*x* es atizador

- 1 $\exists x [Al(x) \wedge B(x)]$
- 2 $\neg\exists x [At(x) \wedge B(x)]$
$\Rightarrow$ $\exists x [At(x) \wedge \neg Al(x)]$

Como el argumento no es correcto buscaremos un contraejemplo, es decir, una interpretación que haga ciertas las premisas y falsa la conclusión. Sea el dominio D={a, b}

x	Al(x)	B(x)	At(x)
a	V	V	F
b	V	F	V

b) **Ningún pájaro, salvo los pavos reales, se pavonea de su cola.**
Algunos pájaros que se pavonean de sus colas no saben cantar.
***Luego,* algunos pavos reales no saben cantar.**

Solución:

P(x)	*x* es un pájaro
PR(x)	*x* es pavo real
PC(x)	*x* se pavonea de su cola
C(x)	*x* sabe cantar

- 1 $\neg\exists x [P(x) \wedge PC(x) \wedge \neg PR(x)]$
- 2 $\exists x [P(x) \wedge PC(x) \wedge \neg C(x)]$
$\Rightarrow$ $\exists x [PR(x) \wedge \neg C(x)]$

Vamos a demostrar, por Deducción Natural, que el argumento es correcto:

- 1	$\neg\exists x [P(x) \wedge PC(x) \wedge \neg PR(x)]$	
- 2	$\exists x [P(x) \wedge PC(x) \wedge \neg C(x)]$	
3	$\forall x \neg[P(x) \wedge PC(x) \wedge \neg PR(x)]$	EN 1
4	$\forall x [\neg(P(x) \wedge PC(x)) \vee \neg\neg PR(x)]$	I, DM 3
5	$P(a) \wedge PC(a) \wedge \neg C(a)$	
6	$\neg(P(a) \wedge PC(a)) \vee \neg\neg PR(a)$	EU 4
7	$P(a) \wedge PC(a)$	EC 5
8	$\neg\neg PR(a)$	SD 6, 7
9	$PR(a)$	EN 8
10	$\neg C(a)$	EC 5
11	$PR(a) \wedge \neg C(a)$	IC 9, 10
12	$\exists x [PR(x) \wedge \neg C(x)]$	IE 11
13	$\exists x [PR(x) \wedge \neg C(x)]$	EE 2, 5-12

c) **Ninguna rana es poética.**
Algunos ánades están desprovistos de poesía.

***Luego*, algunos ánades no son ranas.**

Solución:

R(x)	*x* es rana
P(x)	*x* es poético
A(x)	*x* es un ánade

- 1 $\neg\exists x [R(x) \wedge P(x)]$
- 2 $\exists x [A(x) \wedge \neg P(x)]$
$\Rightarrow$ $\exists x [A(x) \wedge \neg R(x)]$

Como el argumento no es correcto buscaremos un contraejemplo, es decir, una interpretación que haga ciertas las premisas y falsa la conclusión. Sea el dominio D={a}

x	R(x)	P(x)	A(x)
a	V	F	V

7. TRABAJOS COMPLEMENTARIOS

1.- Fundamentación de las Reglas Derivadas para la Deducción Natural en el Cálculo de Proposiciones.

2.- Demostración de las Reglas de Cuantificación del Cálculo de Predicados.

8. LA LÓGICA EN LA VIDA

1. Un conocido filósofo estaba dando una charla sobre lingüística y acababa de decir que la construcción de la doble negación tiene un sentido positivo en algunas lenguas naturales; sin embargo en ninguna lengua se daba el caso de que una construcción con doble afirmación tuviera un sentido negativo. A esto que otro famoso filósofo, que estaba sentado en la parte de atrás de la sala, contestó con tono burlón: "sí, sí".

2. "Cuando mi hijo Arthur tenía diez años, se encaprichó de una escopeta de balines. Me mostré como un padre severo y le dije que no podía ser /.../ Como todos los niños, siguió suplicando e insistiendo hasta que, exasperado, le dije:
   - Mira hijo, mientras sea yo quien mande en esta casa, no tendrás esa escopeta.
   - Papá, si consigo la escopeta, ya no mandarás en esta casa. "

Groucho Marx, "Groucho y Yo"

3. Vamos a proponer un juego de lógica que se tiene que resolver por deducción a partir de una serie de pistas. El objetivo es correlacionar una serie de propiedades que tienen distintos elementos de nuestro Universo del Discurso. La restricción a la que está sujeto este juego es que no pueden tener dos elementos distintos del dominio la misma característica.

Un alumno de Informática, debido al nerviosismo del primer día de clase, sólo recuerda el nombre de sus profesores (María, Jesús y Faraón), las asignatura que se imparten (Lógica, Programación y Matemáticas) y el día de la semana de las distintas clases (lunes, miércoles y jueves). Además recuerda que :

- La clase de Programación, impartida por María, es posterior a la de Lógica.

- A Faraón no le gusta trabajar los lunes, día en el que no se imparte Lógica.

¿ Serías capaz de ayudarle a relacionar cada profesor con su asignatura, así como el día de la semana que se imparte ?

NOTAS: Sabemos que cada profesor imparte una única asignatura, y que las clases se dan en días diferentes. Para resolverlo puedes utilizar el siguiente cuadro de doble entrada.

	Lógica	**Program.**	**Matemát.**	**Lunes**	**Miércoles**	**Jueves**
María						
Jesús						
Faraón						
Lunes						
Miércoles						
Jueves						

DEDUCCIÓN NATURAL EN LÓGICA DE PRIMER ORDEN
REGLAS BÁSICAS

IMPLICACIÓN

TD (Teorema de Deducción)

$$\begin{array}{l}\left[\begin{array}{l}A\\B\end{array}\right.\\\hline A \rightarrow B\end{array}$$

MP (Modus Ponendo Ponens)

$$\begin{array}{l}A \rightarrow B\\A\\\hline B\end{array}$$

CONJUNCIÓN

IC

$$\begin{array}{l}A\\B\\\hline A \wedge B\end{array}$$

EC_1

$$\begin{array}{l}A \wedge B\\\hline A\end{array}$$

EC_2

$$\begin{array}{l}A \wedge B\\\hline B\end{array}$$

DISYUNCIÓN

ID_1

$$\begin{array}{l}A\\\hline A \vee B\end{array}$$

ID_2

$$\begin{array}{l}B\\\hline A \vee B\end{array}$$

ED (Prueba por Casos)

$$\begin{array}{l}A \vee B\\\left[\begin{array}{l}A\\C\end{array}\right.\\\left[\begin{array}{l}B\\C\end{array}\right.\\\hline C\end{array}$$

NEGACIÓN

IN (Reducción al Absurdo)

$$\begin{array}{l}\left[\begin{array}{l}A\\B \wedge \neg B\end{array}\right.\\\hline \neg A\end{array}$$

EN

$$\begin{array}{l}\neg \neg A\\\hline A\end{array}$$

CUANTIFICACIÓN UNIVERSAL

IU (con restricciones)

$$\begin{array}{l}P(a)\\\hline \forall x\ P(x)\end{array}$$

EU

$$\begin{array}{l}\forall x\ P(x)\\\hline P(a)\end{array}$$

CUANTIFICACIÓN EXISTENCIAL

IE

$$\begin{array}{l}P(a)\\\hline \exists x\ P(x)\end{array}$$

EE (con restricciones)

$$\begin{array}{l}\exists x\ P(x)\\\left[\begin{array}{l}P(a)\\B\end{array}\right.\\\hline B\end{array}$$

REGLAS DERIVADAS

REGLAS DE IMPLICACIÓN

Sil (Silogismo Hipotético) **Mut** (Mutación de Premisas) **Id** (Identidad) **CPr** (Carga de Premisa)

$$\frac{A \to B \quad B \to C}{A \to C} \qquad \frac{A \to (B \to C)}{B \to (A \to C)} \qquad \frac{A}{A} \qquad \frac{A}{B \to A}$$

REGLAS DE CONJUNCIÓN Y DISYUNCIÓN

CC y **CD** (Conmutativa) **AC** y **AD** (Asociativa)

$$\frac{A \wedge B}{B \wedge A} \qquad \frac{A \vee B}{B \vee A} \qquad \frac{(A \wedge B) \wedge C}{A \wedge (B \wedge C)} \qquad \frac{(A \vee B) \vee C}{A \vee (B \vee C)}$$

DC (Distributiva de la Conjunción en Disyunción) **DD** (Distributiva de la Disyunción en Conjunción)

$$\frac{A \wedge (B \vee C)}{(A \wedge B) \vee (A \wedge C)} \qquad \frac{A \vee (B \wedge C)}{(A \vee B) \wedge (A \vee C)}$$

IdC y **IdD** (Idempotencia) **AbsC** y **AbsD** (Absorción)

$$\frac{A \wedge A}{A} \qquad \frac{A \vee A}{A} \qquad \frac{A \wedge (A \vee B)}{A} \qquad \frac{A \vee (A \wedge B)}{A}$$

REGLAS DE NEGACIÓN

Cp (Contraposición) **MT** (Modus Tollens) **IDN** (Introducción de Doble Negador)

$$\frac{A \to B}{\neg B \to \neg A} \qquad \frac{A \to B \quad \neg B}{\neg A} \qquad \frac{A}{\neg \neg A}$$

ECQ (Ex Contradictione Quodlibet) **PNC** (P. No Contradicción) **PTE** (P. de Exclusión de tercero)

$$\frac{A \wedge \neg A}{B} \qquad \neg (A \wedge \neg A) \qquad A \vee \neg A$$

REGLAS ADICIONALES

Imp (Importación)

$$\frac{A \rightarrow (B \rightarrow C)}{A \wedge B \rightarrow C}$$

Exp (Exportación)

$$\frac{A \wedge B \rightarrow C}{A \rightarrow (B \rightarrow C)}$$

SD_1 y **SD_2** (Silogismo Disyuntivo)

$$\frac{A \vee B \qquad \neg B}{A} \qquad \frac{A \vee B \qquad \neg A}{B}$$

(Dilemas) **Dil_1**

$$\frac{A \vee B \qquad A \rightarrow C \qquad B \rightarrow C}{C}$$

Dil_2

$$\frac{\neg A \vee \neg B \qquad C \rightarrow A \qquad C \rightarrow B}{\neg C}$$

Dil_3

$$\frac{A \vee B \qquad A \rightarrow C \qquad B \rightarrow D}{C \vee D}$$

Dil_4

$$\frac{\neg A \vee \neg B \qquad C \rightarrow A \qquad D \rightarrow B}{\neg C \vee \neg D}$$

REGLAS DE COIMPLICACIÓN

ICO

$$\frac{A \rightarrow B \qquad B \rightarrow A}{A \leftrightarrow B}$$

ECO_1

$$\frac{A \leftrightarrow B}{A \rightarrow B}$$

ECO_2

$$\frac{A \leftrightarrow B}{B \rightarrow A}$$

INTERCAMBIO

I

$$\frac{A \leftrightarrow B \qquad C_A}{C_B}$$

LEYES DE INTERDEFINICIÓN

(Definición del Implicador) **DI_1**

$$\frac{A \rightarrow B}{\neg (A \wedge \neg B)}$$

DI_2

$$\frac{A \rightarrow B}{\neg A \vee B}$$

(Definición del Conjuntor) **DfC_1** y **DfC_2**

$$\frac{A \wedge B}{\neg(A \rightarrow \neg B)} \qquad \frac{A \wedge B}{\neg (\neg A \vee \neg B)}$$

(Definición del Disyuntor) **DfD_1 y DfD_2**

$$\frac{A \vee B}{\neg A \rightarrow B} \qquad \frac{A \vee B}{\neg(\neg A \wedge \neg B)}$$

LEYES DE DE MORGAN

DM_1

$$\frac{\neg (A \wedge B)}{\neg A \vee \neg B}$$

DM_2

$$\frac{\neg (A \vee B)}{\neg A \wedge \neg B}$$

Capítulo **5**

TEORÍA AXIOMÁTICA

En este capítulo vamos a ver otra forma de realizar deducciones, que aunque menos «natural» que el anterior, nos permite observar cómo a partir de un número limitado de axiomas y una sola regla de inferencia podemos realizar cualquier deducción; es decir, en los axiomas y la regla de inferencia están contenidas implícitamente todas las fórmulas que son correctas en nuestro sistema. Este enfoque está más cercano a la idea de Programación Lógica: base de conocimientos + motor de inferencia.

El método de Deducción Natural tiene la ventaja de la naturalidad, ya que está muy cerca del razonamiento utilizado en matemáticas. Este método tiene como elementos básicos formas del tipo

$$P_1, P_2, \ldots P_n \Rightarrow Q$$

y se construye a partir de una serie de reglas que permiten pasar de unas estructuras deductivas a otras. P_1, P_2, ... P_n son *premisas* que se establecen como válidas en el transcurso de la deducción en curso, pero que una vez establecida dicha deducción dejan de serlo, es decir, no se establecen como válidas en el sistema. Ahora vamos a estudiar un proceso de razonamiento que trabaja con unos enunciados determinados que son aceptados de antemano, según un cierto criterio de racionalidad y según la teoría a tratar. A estos enunciados los llamaremos *Axiomas*.

Axioma : Formatos o esquemas de fórmulas estructuralmente asumidos como válidos o correctos.

Una vez establecidos dichos enunciados, no se van a admitir otros que no sean los que se deduzcan de ellos por inferencia lógica, y estos enunciados se llamarán ***Teoremas***.

Teoría axiomatizada : Teoría deductivamente ordenada en axiomas y teoremas según unas determinadas reglas de inferencia.

Se trabaja ahora con unas estructuras deductivas que tienen los siguientes elementos:

1.- Sistema de *fórmulas válidas*, construido rigurosamente a partir de:
* Esquemas de fórmulas que se asumen como válidos por hipótesis (*axiomas*).
* *Reglas de demostración* o *inferencia* que permiten obtener nuevas fórmulas válidas (*teoremas*), a partir de los axiomas.

2.- Definición de *Deducción* que permite, aplicando las reglas del sistema representar cualquier deducción correcta por una fórmula válida en el sistema formal.

El sistema de axiomas y reglas de demostración deber ser **consistente**, es decir, no debe poder demostrarse una fórmula y su negación.

1. SISTEMA DE KLEENE

El sistema con el que vamos a trabajar será el de Kleene (1953), que es una estructura orientada a la búsqueda de procesos de demostración, aunque su formulación no permite plantear métodos sistemáticos. En la representación de este sistema, introduciremos los axiomas como fórmulas proposicionales válidas, para ello utilizaremos el símbolo metalingüístico |- para indicar que la proposición es una *fórmula válida*.

Sean A, B y C fórmulas bien formadas (fbf) cualesquiera.

- Los *axiomas* son:

A.1	$\vdash A \rightarrow (B \rightarrow A)$	
A.2	$\vdash (A \rightarrow B) \rightarrow [(A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow C)]$	
A.3	$\vdash A \rightarrow (B \rightarrow A \wedge B)$	
A.4	$\vdash A \wedge B \rightarrow A$	$\vdash A \wedge B \rightarrow B$
A.5	$\vdash A \rightarrow A \vee B$	$\vdash B \rightarrow A \vee B$
A.6	$\vdash (A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow (A \vee B \rightarrow C))$	
A.7	$\vdash (A \rightarrow B) \rightarrow ((A \rightarrow \neg B) \rightarrow \neg A))$	
A.8	$\vdash \neg \neg A \rightarrow A$	

A.9 $\vdash \forall x\, B(x) \rightarrow B(a)$
A.10 $\vdash B(a) \rightarrow \exists x\, B(x)$

Estos 10 axiomas se asumen como válidos en el sistema, es decir cualquier fórmula bien formada que tenga una forma equivalente a uno de ellos es una fórmula válida por hipótesis.

Ejemplo:

$p \wedge q \rightarrow (r \rightarrow p \wedge q)$ es una fórmula proposicional con estructura equivalente al A.1, por tanto no es necesario demostrar que es válida, lo es ya por hipótesis.

- *Reglas básicas de demostración*: tendremos una regla para la inferencia de nuevas reglas, en general, y dos para la manipulación de los cuantificadores.

  - La denominada en lógica clásica «modus ponens» (MP):

    $\vdash A$
    $\vdash A \rightarrow B$
    $\vdash B$

  - Introducción Universal Condicional (IUC)

    $\vdash A \rightarrow B(a)$
    $\vdash A \rightarrow \forall x\, B(x)$ (en *A* no *a*, y *A* fórmula cualquiera)

  - Introducción Existencial Condicional (IEC)

    $\vdash A(a) \rightarrow B$
    $\vdash \exists x\, A(x) \rightarrow B$ (no *a* en *B*, y *B* fórmula cualquiera)

En deducción axiomática, estas son las únicas reglas de inferencia que se pueden utilizar; las reglas que hemos visto en deducción natural, se podrán utilizar no como reglas sino como estructuras de fórmulas válidas; es decir que se debe de obtener una fórmula válida de la regla que se quiera utilizar, tanto si la regla es básica como derivada. Una vez establecida de esta forma, ya si puede entrar a formar parte de la deducción axiomática. Esto es posible gracias al Teorema de Deducción que veremos en el siguiente tema, y que permite pasar de estructuras deductivas a fórmulas válida, y viceversa.

Deducción : Una lista finita de fórmulas (A_1, A_2, ..., A_n) es una deducción A_n a partir de las hipótesis H_1, H_2, ..., H_m ($0 \leq m \leq n$) si cada fórmula A_i de la lista es:

1.- A_i es uno de los axiomas, o
2.- A_i es una de las hipótesis H_1, ..., H_m, o
3.- A_i es una fórmula que resulta de dos fórmulas, A_h y A_k ($h < i$, $k < i$), por aplicación de alguna de las reglas de inferencia.

A_n se llama *fórmula deducida* de las hipótesis consideradas.

Demostración : Una demostración de una fórmula A en el sistema axiomático, es una sucesión de fórmulas A_1, A_2, ..., A_n tal que, para todo i ($1 \leq i \leq n$) se verifica:
1.- A_i es un axioma del sistema, o
2.- A_i se deduce de dos miembros anteriores, A_h y A_k ($h < i$, $k < i$), como consecuencia directa aplicando las reglas de inferencia.

Se dirá entonces que A_n es un *teorema* del sistema.

Conclusiones:

1. Si A_1, ..., A_n es demostrable y $k < n$, entonces A_1, ..., A_k es demostrable; luego A_k, es también un teorema.
2. Los axiomas del sistema son teoremas, porque sus elementos son sucesiones con un solo elemento.
3. Una demostración es una deducción sin hipótesis, por tanto la fórmula deducida sólo es válida si se dan las hipótesis, mientras que el teorema lo es siempre en el sistema.

Ejemplo:

Demostrar |- $A \rightarrow A$, o sea que la fórmula $A \rightarrow A$ es válida. Es un caso de demostración ya que no tenemos premisas iniciales.

1 |- $A \rightarrow (A \rightarrow A)$	A.1 $(B \Leftrightarrow A)$
2 |- $(A \rightarrow (A \rightarrow A)) \rightarrow$ $\rightarrow ((A \rightarrow ((A \rightarrow A) \rightarrow A)) \rightarrow (A \rightarrow A))$	A.2 $(B \Leftrightarrow A \rightarrow A)$ $(C \Leftrightarrow A)$
3 |- $(A \rightarrow ((A \rightarrow A) \rightarrow A)) \rightarrow (A \rightarrow A)$	MP 1, 2
4 |- $A \rightarrow ((A \rightarrow A) \rightarrow A)$	A.1 $(B \Leftrightarrow A \rightarrow A)$
5 |- $A \rightarrow A$	MP 3, 4

Podemos decir que $A \rightarrow A$ es un teorema (**Teorema de identidad**), y por tanto es una fórmula válida.

Hemos construido un sistema formal en el cual ciertas fbf son teoremas, y esto se consigue haciendo una sucesión finita que constituya una demostración.

2. OBTENCIÓN DE TEOREMAS

Vamos a estudiar que pasos debemos seguir para obtener los teoremas de nuestro sistema. En general podemos seguir el siguiente procedimiento:

1.- Si la fórmula problema guarda ya relación de identidad formal con alguna de las fórmulas o esquemas de fórmulas ya admitidos, se admite sin más.

2.- Si no es viable, se hará coincidir mediante cambios oportunos (sustituciones) con las fórmulas del sistema.

3.- Para simplificar o hacer más fácil la demostración, utilizar teoremas ya demostrados.

4.- La principal herramienta es el «Teorema de Deducción».

2.1. Teorema de Deducción

En este tema vamos únicamente a enunciar el teorema para poder utilizarlo en nuestras demostraciones. Ya se tratará con detalle en el proximo capítulo.

Si es posible deducir una fórmula B de una serie de fórmulas F (que puede ser vacía), y de una fórmula A, entonces es posible deducir también la fórmula $A \rightarrow B$ de la referida serie de fórmulas F.

$$\text{Si } F, A \Rightarrow B \text{ entonces } F \Rightarrow A \rightarrow B$$

El reciproco del Teorema de Deducción también existe:

$$\text{Si } F \Rightarrow A \rightarrow B \text{ entonces } F, A \Rightarrow B$$

Corolario: Dada una estructura deductiva correcta siempre es posible encontrar una fórmula válida que la represente.

Ejemplo:
Sea la estructura deductiva:

$$p_1, p_2, p_3 \Rightarrow q$$

por aplicación sucesiva del «Teorema de Deducción», son correctas las siguientes deducciones:

$$p_1, p_2 \Rightarrow p_3 \rightarrow q$$

$$p_1 \Rightarrow p_2 \rightarrow (p_3 \rightarrow q)$$

$$\Rightarrow p_1 \rightarrow (p_2 \rightarrow (p_3 \rightarrow q))$$

Finalmente, esta es la fórmula válida que representa la estructura deductiva citada:

$$\vdash p_1 \rightarrow (p_2 \rightarrow (p_3 \rightarrow q))$$

También, si aplicamos la interdefinición del implicador 3 veces a esta fórmula obtenemos:

$$\neg p_1 \vee (\neg p_2 \vee (\neg p_3 \vee q))$$

y aplicando la asociativa del disyuntor y De Morgan

$$\neg(p_1 \wedge p_2 \wedge p_3) \vee q$$

Por último, aplicando nuevamente la interdefinición del implicador, pero en el sentido inverso, obtenemos una fórmula equivalente a la anterior, y por tanto también representa a la estructura deductiva

$$\vdash p_1 \wedge p_2 \wedge p_3 \rightarrow q$$

Si una deducción es correcta entonces la fórmula asociada a ella, cualquiera de las dos, es tautología, y viceversa.

2.2. Aplicación del Teorema de Deducción

El Teorema de Deducción nos facilita la obtención de teoremas porque cuando hay que demostrar una implicación, podemos empezar suponiendo los antecedentes y dedicarnos sin más a la búsqueda del consecuente. Una vez conseguido éste, por el Teorema de Deducción aplicado sucesivas veces, obtenemos la implicación deseada.

De esta manera, podemos ver como los 10 axiomas del Sistema de Kleene tienen su correspondencia con reglas vistas de la Deducción Natural:

A.1	Carga de premisa
A.2	Silogismo hipotético
A.3	Introducción de la conjunción
A.4	Eliminación de la conjunción
A.5	Introducción de la disyunción
A.6	Prueba por casos
A.7	Reducción al absurdo
A.8	Eliminación del negador
A.9	Eliminación del universal
A.10	Introducción del existencial

La formulación de un sistema axiomático puede plantearse con la finalidad puramente matemática de limitar el número de axiomas y reglas de inferencia; pero la que a nosotros, como informáticos, nos interesa es la de servir de definición implícita del conjunto de fórmulas válidas, como vehículo de métodos sistemáticos de búsqueda de demostraciones. Este enfoque conecta directamente con la idea de *Programación Lógica*, a la cual dedicamos un capítulo completo.

3. EJERCICIOS

Comprobar que las siguientes fórmulas son válidas en el sistema de Kleene, utilizando si se cree necesario el «Teorema de Deducción».

1.- |- $\mathbf{(A \to B) \to ((B \to C) \to (A \to C))}$ (Silogismo hipotético)

Solución:

Aplicando el Teorema de Deducción (3 veces), bastará deducir lo siguiente:

1ª	$A \to B \Rightarrow (B \to C) \to (A \to C)$	
2ª	$A \to B,\ B \to C \Rightarrow A \to C$	
3ª	$A \to B,\ B \to C,\ A \Rightarrow C$	
1	$A \to B$	Premisa
2	$B \to C$	Premisa
3	A	Premisa
4	B	MP 1, 3
5	C	MP 2, 4

2.- |- $\mathbf{A \rightarrow (\neg A \rightarrow C)}$ (ECQ)

Solución:

Por el Teorema de Deducción basta comprobar que es correcta la deducción:

$A, \neg A \Rightarrow C$

1	A	Premisa
2	¬A	Premisa
3	$\vdash A \rightarrow (\neg C \rightarrow A)$	A.1
4	$\neg C \rightarrow A$	MP 1, 3
5	$\vdash \neg A \rightarrow (\neg C \rightarrow \neg A)$	A.1
6	$\neg C \rightarrow \neg A$	MP 2, 5
7	$\vdash (\neg C \rightarrow A) \rightarrow [(\neg C \rightarrow \neg A) \rightarrow \neg\neg C]$	A.7
8	$(\neg C \rightarrow \neg A) \rightarrow \neg\neg C$	MP 4, 7
9	$\neg\neg C$	MP 6, 8
10	$\neg\neg C \rightarrow C$	A.8
11	C	MP 9, 10

3.- |- $\mathbf{(A \rightarrow B) \rightarrow \neg A \vee B}$ (Interdefinición de Conectivas)

Solución:

Sin utilizar el Teorema de Deducción.

1	$\vdash \neg A \rightarrow \neg A \vee B$	A.5
2	$\vdash B \rightarrow \neg A \vee B$	A.5
3	$\vdash \neg\neg B \rightarrow B$	A.8
4	$\vdash (\neg\neg B \rightarrow B) \rightarrow ((B \rightarrow \neg A \vee B) \rightarrow (\neg\neg B \rightarrow \neg A \vee B))$	T. Sil.
5	$\vdash (B \rightarrow \neg A \vee B) \rightarrow (\neg\neg B \rightarrow \neg A \vee B)$	MP 3, 4
6	$\vdash \neg\neg B \rightarrow \neg A \vee B$	MP 2, 5
7	$\vdash (\neg A \rightarrow \neg A \vee B) \rightarrow \rightarrow[(\neg\neg B \rightarrow \neg A \vee B) \rightarrow (\neg A \vee \neg\neg B \rightarrow \neg A \vee B)]$	A.6
8	$\vdash (\neg\neg B \rightarrow \neg A \vee B) \rightarrow (\neg A \vee \neg\neg B \rightarrow \neg A \vee B)$	MP 1, 7
9	$\neg A \vee \neg\neg B \rightarrow \neg A \vee B$	MP 6, 8
10	$\vdash \neg(A \wedge \neg B) \rightarrow \neg A \vee \neg\neg B$	Leyes De Morgan
11	$\vdash (\neg(A \wedge \neg B) \rightarrow \neg A \vee \neg\neg B) \rightarrow [(\neg A \vee \neg\neg B \rightarrow \neg A \vee B) \rightarrow \rightarrow (\neg(A \wedge \neg B) \rightarrow \neg A \vee \neg\neg B)]$	T. Sil.
12	$\vdash (\neg A \vee \neg\neg B \rightarrow \neg A \vee B) \rightarrow (\neg(A \wedge \neg B) \rightarrow \neg A \vee \neg\neg B)$	MP 10, 11
13	$\vdash \neg(A \wedge \neg B) \rightarrow \neg A \vee B$	MP 9, 12
14	$\vdash (A \rightarrow B) \rightarrow \neg(A \wedge \neg B)$	T. Interdef.
15	$\vdash ((A \rightarrow B) \rightarrow \neg(A \wedge \neg B)) \rightarrow [(\neg(A \wedge \neg B) \rightarrow \neg A \vee B) \rightarrow \rightarrow ((A \rightarrow B) \rightarrow \neg A \vee B)]$	T. Sil.
16	$\vdash (\neg(A \wedge \neg B) \rightarrow \neg A \vee B) \rightarrow ((A \rightarrow B) \rightarrow \neg A \vee B)$	MP 14, 15
17	$\vdash (A \rightarrow B) \rightarrow \neg A \vee B$	MP 13, 16

4.- $\vdash \neg\forall x\ P(x) \leftrightarrow \exists x\ \neg P(x)$ (Negador de Universal)

Solución:

1º $\vdash \neg\forall x\ P(x) \rightarrow \exists x\ \neg P(x)$

1	$\vdash \neg P(a) \rightarrow \exists x\ \neg P(x)$	A.10
2	$\vdash (\neg P(y) \rightarrow \exists x \neg P(x)) \rightarrow$ $\rightarrow (\neg\exists x\ \neg P(x) \rightarrow P(y))$	Cp
3	$\vdash \neg\exists x\ \neg P(x) \rightarrow P(y)$	MP 1, 2
4	$\vdash \neg\exists x \neg P(x) \rightarrow \forall x\ P(x)$	IUC 3
5	$\vdash (\neg\exists x \neg P(x) \rightarrow \forall x\ P(x)) \rightarrow$ $\rightarrow (\neg\forall x\ P(x) \rightarrow \exists x \neg P(x))$	Cp
6	$\vdash \neg\forall x\ P(x) \rightarrow \exists x \neg P(x)$	MP 4, 5

2º $\vdash \exists x\ \neg P(x) \rightarrow \neg\forall x\ P(x)$

1	$\vdash \forall x\ P(x) \rightarrow P(a)$	A.9
2	$\vdash (\forall x\ P(x) \rightarrow P(a)) \rightarrow$ $\rightarrow (\neg P(a) \rightarrow \neg\forall x\ P(x))$	Cp
3	$\vdash \neg P(a) \rightarrow \neg\forall x\ P(x)$	MP 1, 2
4	$\vdash \exists x \neg P(x) \rightarrow \neg\forall x\ P(x)$	IEC 3

4. TRABAJOS COMPLEMENTARIOS

Realizar la demostración de los distintos Teoremas correspondientes a las Reglas Derivadas vistas en la Deducción Natural.

5. LA LÓGICA EN LA VIDA

1. Hay una antigua anécdota sobre un filósofo sofista, Protágoras, que aceptó instruir a Euatlo en retórica para que éste pudiera practicar la abogacía. Euatlo a su vez convino en pagar a Protágoras sus honorarios sólo después de ganar su primer pleito. Euatlo, sin embargo, decidió no practicar la abogacía el terminar su formación y entonces Protágoras lo demandó para obtener sus honorarios. Protágoras sostenía que Euatlo tenía que pagarle de todos modos: si él ganaba el pleito, debería ser pagado por mandato judicial; si lo perdía, debería ser pagado en razón a los términos de su acuerdo con Euatlo. Euatlo, que había aprendido algo de su estudio, sostenía que no tenía que pagar de ningún modo: si ganaba el pleito no tendría que pagar por

mandato judicial; si lo perdía no tendría que pagar en razón a los términos de su acuerdo con Protágoras.

2. "Como dije, el juicio fue muy complicado. El primer sospechoso era la Sota de Corazones, pero se presentó una prueba circunstancial que estableció sin la menor duda que la Sota de Corazones no pudo haber robado los pasteles. El siguiente sospechoso era el Lirón. Pero varios testigos de confianza declararon que el Lirón estaba profundamente dormido a la hora del robo, por lo tanto no pudo haber sido él. Aquí el juicio quedó paralizado.

De repente se abrió de golpe la puerta de la sala y entró con orgullo el Conejo Blanco llevando la bandeja de los pasteles. Tras él entraron los soldados con el Grifo y la Falsa Tortuga encadenados.

- Los pasteles estaban en la playa - explicó el Conejo Blanco -. El Grifo y la Tortuga estaban a punto de comérselos cuando los soldados pasaron por allí, y los detuvieron al instante.

- Eso demuestra su culpabilidad sin la menor duda - gritó la Reina -, así que ¡a dejarles sin cabeza inmediatamente!

- Bueno, bueno - dijo el Rey -, hay que hacerles un juicio justo.

Sucedieron cosas que demostraron que el Grifo y la Tortuga no eran ambos culpables, lo que quedaba por aclarar era si lo era uno de los dos, y en ese caso, cuál; o si algún otro era culpable: ¿Fue una mera coincidencia el que los pasteles fueran encontrados por el Grifo y la Tortuga? No; pronto se presentaron pruebas que demostraron de manera concluyente que o bien el Grifo o bien la Falsa Tortuga era el culpable (pero no ambos), pero el tribunal no encontró la manera de decidir quién lo era. Cuando parecía que no podía avanzarse más, apareció una mezcolanza de testigos que hicieron diversas declaraciones.

- El Grifo no robó los pasteles - dijo la Duquesa.

- Pero ha robado otras cosas con anterioridad - dijo la Cocinera.

- La Tortuga ha robado cosas antes - dijo el Gato de Cheshire.

- El Gato de Cheshire ha robado cosas alguna vez - dijo la Oruga.

- La Cocinera y el Gato de Cheshire tienen ambos razón - dijo la Liebre de Marzo.

- La Cocinera y la Oruga tienen ambas razón - dijo el Lirón.

- O bien tiene razón el Gato de Cheshire o la tiene la Oruga, o puede que ambos - dijo el Sombrerero.

- O bien tiene razón la Liebre o la tiene el Lirón, o puede que ambos - dijo Bill el Lagarto.

- La Cocinera y el Sombrerero tienen ambos razón - dijo la Sota de Corazones.

- Bill el Lagarto tiene razón y la Sota de Corazones no la tiene - dijo el Conejo Blanco.

Se produjo un gran silencio.

- ¡ Todo esto no demuestra nada ! - rugió el Rey - Sólo palabras, palabras, palabras inútiles.

- No tan inútiles, Majestad - dijo Alicia desde el jurado, poniéndose en pie -. Resulta que el Conejo Blanco y la Duquesa hicieron declaraciones que o bien ambas son ciertas o bien ambas son falsas.

Todos los ojos se volvieron ansiosos hacia Alicia. Todo el mundo sabía que Alicia sólo hace declaraciones ciertas, y una investigación posterior demostró que esta declaración no era una excepción. Más aún, esta declaración resolvió todo el misterio.

¿ Quién robó los pasteles ? ”

Raymond Smullyan
“*Alicia en el país de las adivinanzas*”

DEDUCCIÓN AXIOMÁTICA EN LÓGICA DE PRIMER ORDEN

Sistema de KLEENE

* Los *axiomas* son:

A.1 |- $A \rightarrow (B \rightarrow A)$

A.2 |- $(A \rightarrow B) \rightarrow [(A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow C)]$

A.3 |- $A \rightarrow (B \rightarrow A \wedge B)$

A.4 |- $A \wedge B \rightarrow A$ |- $A \wedge B \rightarrow B$

A.5 |- $A \rightarrow A \vee B$ |- $B \rightarrow A \vee B$

A.6 |- $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow (A \vee B \rightarrow C))$

A.7 |- $(A \rightarrow B) \rightarrow ((A \rightarrow \neg B) \rightarrow \neg A))$

A.8 |- $\neg \neg A \rightarrow A$

A.9 |- $\forall x B(x) \rightarrow B(a)$

A.10 |- $B(a) \rightarrow \exists x B(x)$

* *Reglas básicas de demostración*:

“Modus Ponens” (MP)

|- A
|- $A \rightarrow B$
|- B

Introducción Universal Condicional (IUC)

|- $A \rightarrow B(a)$
|- $A \rightarrow \forall x B(x)$ (en *A* no *a*, y *A* fórmula cualquiera)

Introducción Existencial Condicional (IEC)

|- $A(a) \rightarrow B$
|- $\exists x A(x) \rightarrow B$ (no *a* en *B*, y *B* fórmula cualquiera)

Capítulo **6**

METALÓGICA

En este capítulo vamos a desarrollar el Teorema de Deducción, que sirve de puente entre los conceptos de deducción correcta y fórmula válida. La Metalógica es la disciplina que estudia los cálculos o lenguajes lógicos, y desarrolla propiedades de los sistemas formales como la completud, consistencia, decibilidad, independencia de los axiomas, etc. Mediante las propiedades de la metalógica, demostraremos en cada uno de los sistemas definidos que una fórmula es tautología si y solo si es válida, o dicho de otra forma que una fórmula es semánticamente válida en teoría de la interpretación si y solo si es formalmente válida en teoría de la demostración.

La metateoría es una disciplina que se ocupa del estudio de las propiedades de un sistema formal por medio de un metalenguaje. La metalógica es la disciplina que estudia y desarrolla propiedades de los sistemas lógicos. En este tema, aparte del Terorema de Deducción, estudiaremos el comportamiento de tres propiedades fundamentales de la metalógica (*correción*, *completud* y *decisión*), tanto para el Cálculo de Proposiciones como para el Cálculo de Predicados.

1. TEOREMA DE DEDUCCIÓN

(**Teorema fundamental de la lógica matemática. Herbrand 1928**)

Enunciado:

Si es posible deducir una fórmula B de una serie de fórmulas F (que puede ser vacía), y de una fórmula A, entonces es posible deducir también la fórmula $A \rightarrow B$ de la referida serie de fórmulas F.

Si $F, A \Rightarrow B$ entonces $F \Rightarrow A \rightarrow B$

Demostración:

La demostración es por inducción sobre el número de FBFs de la sucesión que constituye la deducción de *B* a partir de *F* y de *A*.

Sea V_n la sucesión que obtiene *B* de *F* y de *A*, es decir, $F, A \Rightarrow B$, $n \geq 1$, con *n* el número de líneas (finito) de la referida deducción que se supone ya dada.

Llamaremos *W* la deducción que queremos demostrar ($F \Rightarrow A \rightarrow B$). Demostraremos que existe *W* siempre, siendo *n* cualquier número de líneas de derivación.

Base de la inducción:

n = 1: sucesión V_n de un sólo elemento, es decir, una línea de derivación. Luego esta línea tiene que ser la propia *B*. Así que *B* , por la definición de deducción, es:

1.- un axioma, o
2.- una fórmula de *F*, o
3.- la fórmula *A*.

CASO 1.- Si *B* es un axioma entonces construimos la sucesión *W* a partir de la V_n (*n*=1), es decir, construimos $A \rightarrow B$ a partir de *F*:

1	|- B	(axioma)
2	|- B $\rightarrow$ (A $\rightarrow$ B)	A.1
3	|- A $\rightarrow$ B	MP 1, 2

CASO 2.- Si *B* es una fórmula de *F*, construimos *W*:

1	|- B	(miembro de *F*: una de las hipótesis)
2	|- B $\rightarrow$ (A $\rightarrow$ B)	A.1
3	|- A $\rightarrow$ B	MP 1, 2

CASO 3.- Si *B* es la fórmula *A* entonces hay que demostrar $A \rightarrow A$. Esto se corresponde con el «Teorema de identidad» ya demostrado, luego $\vdash A \rightarrow A$ es fórmula válida y la deducción es la demostración del teorema.

Por tanto, en cualquiera de los tres casos hemos obtenido $F \Rightarrow A \rightarrow B$. Queda, por tanto, completado el paso base.

Paso de inducción:

Suponemos ahora la deducción V_n con *n* líneas de derivación ($n > 1$), y que para cualquier deducción dada A_k ($k < n$) existe correlativamente una deducción subsiguiente *W* ($F \Rightarrow A \rightarrow B$). Entonces ahora se pueden considerar 4 casos:

1.- *B* es un axioma, o
2.- *B* es miembro de *F*, o
3.- *B* es *A*, o
4.- *B* se obtiene de dos FBFs anteriores por aplicación de la regla de demostración MP.

Los casos 1, 2 y 3 son similares al caso base.

CASO 4.- Si *B* se obtiene de dos FBFs anteriores por aplicación de MP, estas dos fórmulas de las que he deducido *B* tendrán la forma *C* y $C \rightarrow B$ (al aplicar MP se obtiene *B*) y cada una de ellas puede deducirse de *F* y de *A* mediante una sucesión de menos de *n* elementos. Tenemos:

$F, A \Rightarrow C$ y
$F, A \Rightarrow C \rightarrow B$

luego por hipótesis de inducción (menos de *n* líneas de derivación) también se dan:

$F \Rightarrow A \rightarrow C$ y
$F \Rightarrow A \rightarrow (C \rightarrow B)$

respectivamente. Luego la sucesión V_n requerida se construye así:

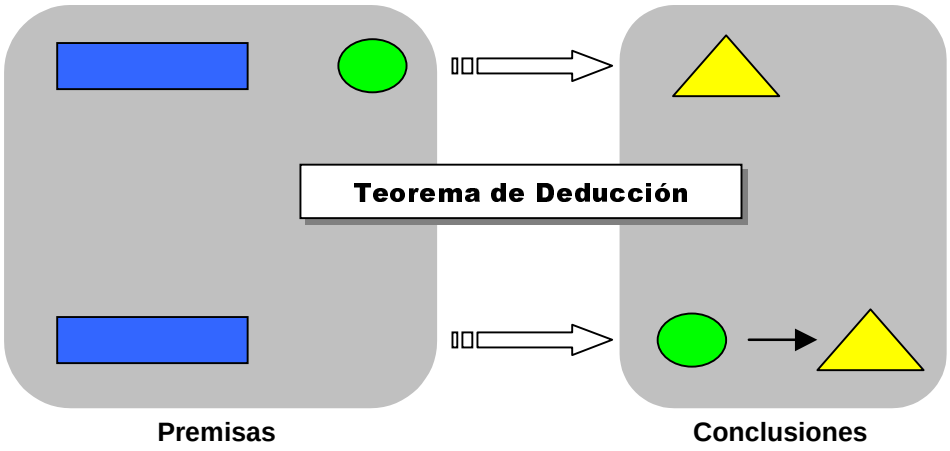
(1)		
.		
.	(deducción de A $\rightarrow$ C a partir de F)	
.		
(k)	|- A $\rightarrow$ C	
(k + 1)		
.		
.	(deducción de A $\rightarrow$ (C $\rightarrow$ B) a partir de F)	
.		
(l)	|- A $\rightarrow$ (C $\rightarrow$ B)	
(l + 1)	|- (A $\rightarrow$ C) $\rightarrow$ [(A $\rightarrow$ (C $\rightarrow$ B)) $\rightarrow$ (A $\rightarrow$ B)]	A.2
(l + 2)	|- (A $\rightarrow$ (C $\rightarrow$ B)) $\rightarrow$ (A $\rightarrow$ B)	MP k, l +1
(l + 3)	|- A $\rightarrow$ B	MP l, l + 2

Luego $F \Rightarrow A \rightarrow B$ en los cuatro casos.

Así pues, por el principio de inducción matemática la proposición se verifica cualquiera que sea el número de FBFs de la deducción *B* a partir de *F* y de *A*. ⌘

NOTA: El reciproco del Teorema de Deducción también existe, y es fácil de demostrar.

Tal como podemos observar en la representación gráfica del Teorema de Deducción que aparece a continuación, podemos considerarlo como una trasmisión de fórmulas de las premisas a la conclusión:



2. PROPIEDADES FORMALES

Los sistemas formales pueden cumplir una serie de propiedades. Para cada uno de los formalismos que hemos desarrollado, Cálculo de Proposiciones y Cálculo de Predicados, veremos qué propiedades verifican cada uno de ellos. Las propiedades a estudiar en un Sistema Formal son:

Corrección: Un sistema formal es consistente, correcto o no contradictorio cuando a través de sus reglas de inferencias y de sus axiomas, no podemos demostrar que sean correctas en el sistema una fórmula y su negación.

Hablamos, por tanto, de sistema correcto en el sentido de que toda sentencia que es demostrable en el sistema es ciertamente una sentencia verdadera, y toda sentencia que es refutable en el sistema es falsa. Dicho de una manera más formal, toda fórmula deducible con el método de cálculo es válida según la noción de verdad.

" Si una fórmula A es formalmente deducible en el sistema, entonces es lógicamente verdadera"

Si |- A entonces |=A

Completo: Toda fórmula que sea semánticamente verdadera, es demostrable con el método de cálculo.

Decimos que el sistema es completo en el sentido de que todas las sentencias verdaderas son demostrables y que todas las falsas son refutables.

" Si una fórmula A es lógicamente verdadera, entonces es formalmente deducible en el sistema"

Si |= A entonces |- A

La conjunción de ambos resultados nos lleva a la coincidencia entre la sintaxis y la semántica:

|- A sii |= A

Decidible: Existe un algoritmo o procedimiento finito que permite determinar si una fórmula es o no válida en el sistema.

Habitualmente diremos que es semidecidible cuando exista un procedimiento para probar que una fórmula es válida, pero no se pueda garantizar que finalice cuando la fórmula no sea válida.

Monótono: Un sistema formal es monótono cuando si una expresión *Q* es un teorema con respecto a un cierto conjunto de axiomas *S*, si añadimos algún axioma nuevo, dicha expresión seguirá siendo un teorema.

Los sistemas lógicos que hemos visto son monótonos en el sentido de que conforme se añaden nuevos axiomas, aparecen nuevas fórmulas bien formadas, pero nunca se vuelven inválidos los resultados anteriores, ya que de hecho, las demostraciones las podemos seguir haciendo usando exclusivamente los axiomas iniciales y no utilizando los nuevos. Sin embargo el concepto de monotonía no lo cumple nuestra forma natural de pensar, por lo que se han desarrollado nuevos tipos de lógicas llamadas *no monótonas* que nos permiten razonar con modelos incompletos o cambiantes.

- El Cálculo de Proposiciones es:
  Completo
  Correcto
  Decidible
  Monótono

- El Cálculo de Predicados es:
  Completo
  Correcto
  No decidible (semidecidible)
  Monótono

3. METALÓGICA DEL CÁLCULO DE PROPOSICIONES

La metalógica del cálculo de proposiciones demostrará que el sistema es consistente, completo y decidible.

3.1. Consistencia

Decimos que un sistema formal es consistente si todo teorema (fórmula) demostrado en él, es verdadero, de tal forma que no pueda darse el caso de la demostración de *A* y de ¬*A* simultáneamente; si |-*A es una fórmula válida entonces A debe ser una tautología* (Teorema de Post).

En el cálculo de proposiciones el sistema debe ser capaz de impedir, mediante el conjunto de axiomas y reglas de inferencia que posee, la demostración de una fórmula y de su negación.

Teorema de Post :

> *" si |-A es una fórmula demostrable entonces |= A es una fórmula válida semánticamente"*

Demostración:

Tenemos que demostrar que la deducibilidad está relacionada con la forma semántica y para ello tenemos que ver que tanto los axiomas como los teoremas son tautologías:

1º.- los axiomas (válidos en teoría de la demostración) son tautologías (son semánticamente válidos).

2º.- los teoremas (válidos en teoría de la demostración) son tautologías (son semánticamente válidos).

1º.- Comprobar que todos los axiomas del cálculo de proposiciones son tautologías es fácil y no costoso ya que podemos ir cogiendo cada uno y utilizar un procedimiento mecánico como por ejemplo el de las tablas de verdad y comprobar la tautología.

Axioma 1: $\vdash A \rightarrow (B \rightarrow A)$

A	B	$B \rightarrow A$	$A \rightarrow (B \rightarrow A)$
V	V	V	V
V	F	V	V
F	V	F	V
F	F	V	V

De esta forma podemos demostrar que todos los axiomas son tautologías.

2º.- Si queremos comprobar que los teoremas son tautologías no podemos hacer lo mismo ya que nos resultaría imposible de realizar, entonces se hace lo siguiente:

Se prueba que las reglas de inferencia del sistema que se aplican al mismo para conseguir los teoremas transmiten la propiedad de la tautología.

En el cálculo de proposiciones la regla de inferencia del sistema axiomático es la del modus ponens, si conseguimos demostrar que esta fórmula es tautología tendremos demostrado que todos los teoremas que se deduzcan de los axiomas y de aplicación de esta regla serán también tautología, es decir, el modus ponens obtiene tautología en la conclusión cuando las premisas a las que se aplica son también tautologías.

Aplicamos el siguiente resultado:

Una estructura deductiva $P_1, P_2, ... P_n \Rightarrow Q$ es correcta, cuando no se da el caso de que las premisas $P_1, P_2, ...P_n$ sean verdaderas y la conclusión Q sea falsa.

El modus ponnes tiene la siguiente estructura deductiva:

$$A, A \rightarrow B \Rightarrow B$$

mediante las tablas de verdad, vemos que es tautología:

A	B	$A \rightarrow B$
V	**V**	**V**
V	F	F
F	V	V
F	F	V

No se da el caso de que las premisas sean verdaderas y la conclusión falsa, luego queda demostrado que la regla del modus ponens transmite la herencia de la tautología a sus premisas. Entonces, todo teorema que se deduzca de la aplicación de dicha regla de inferencia y de los axiomas también será tautología.

c.q.d.

3.2. Completud

Decimos que un sistema es completo si todas las verdades del sistema pueden ser representadas como teoremas demostrables en él, es decir, si todas las fórmulas que son semánticamente válidas en el sistema, son teoremas. Esta propiedad se conoce como teorema de Kalmar (1934 - 1935), y para demostrarla vamos a utilizar primero un lema que relaciona el concepto semántico con el sintáctico de deducibilidad, mediante deducciones asociadas a las líneas de una tabla de verdad.

Lema de Kalmar

Cualquier fórmula *S* tiene asociada un conjunto de deducciones correctas en teoría de la demostración, exactamente una por cada línea de la tabla de verdad, construida mediante una función $D_m(P_i)$ de las proposiciones componentes P_i, de acuerdo con las reglas siguientes:

Si en la línea 'm' de la tabla de verdad aparece la proposición con significado verdadero, la deducción correspondiente es

$$D_m (P_i) = P_i$$

y si aparece con significado F, la deducción es:

$$D_m (P_i) = \neg P_i$$

es decir, que si tengo un tabla de verdad puedo representar cada línea de dicha tabla por su correspondiente deducción

Teorema de KALMAR

Si una fórmula S se define a partir de las proposiciones P_1, P_2, ... P_n y es una tautología, la fórmula S es válida, es decir:
$\models S$ *entonces* $\vdash S$

Demostración:

Por hipótesis *S* es tautología, entonces por definición, tenemos que *S* es verdadera para cada atribución veritativa. y si *S* es verdadera para cada una de sus atribuciones es porque es deducible de cada una de ellas, sea cual sea el valor de verdad de cada una de ellas, es decir, según el lema anterior, las deducciones asociadas a cada línea de la tabla de verdad son:

$P_1,\ P_2,\ ....\quad P_n \quad \Rightarrow S$
$\neg P_1, P_2,\ ....\quad P_n \quad \Rightarrow S$
$P_1, \neg P_2,\ ....\quad P_n \quad \Rightarrow S$
$\neg P_1, \neg P_2,\ ....\quad P_n \quad \Rightarrow S$
$\neg P_1, P_2, \neg P_3,\ ....\quad P_n \quad \Rightarrow S$
$\neg P_1, \neg P_2, \neg P_3,\ ...\quad P_n \quad \Rightarrow S$
.....
.....
$\neg P_1, \neg P_2, \neg P_3,\ ...\quad \neg P_n \quad \Rightarrow S$

Luego si tenemos *n* proposiciones tenemos 2^n estructuras deductivas correspondientes a la aparición de P_i, o de $\neg P_i$. es decir, puedo formar 2^{n-1} pares de estructuras donde el primer par corresponde al valor P_n = V y Γ_i todas las posibles combinaciones de asignaciones veritativas de las P_{n-1}, y similar el segundo par :

$P_n , \Gamma_i \Rightarrow S \qquad\qquad \neg P_n , \Gamma_i \Rightarrow S$

A partir de cada pareja, se obtiene por deducción

$P_n \vee \neg P_n , \Gamma \Rightarrow S$ (aplicando la regla de disyunción).

Si volvemos a aplicar el proceso anterior junto con la deducción, obtendremos 2^{n-2} estructuras :

$$P_n \vee \neg P_n ,\ P_{n-1} \vee \neg P_{n-1} , \Gamma_j \Rightarrow S,$$

y si aplicamos sucesivamente este proceso, tenemos:

$$P_n \vee \neg P_n ,\ P_{n-1} \vee \neg P_{n-1},.....\ P_1 \vee \neg P_1 \Rightarrow S$$

si aplicamos el teorema de deducción, *n* veces, a la estructura anterior, nos queda:

$P_n \vee \neg P_n ,\ P_{n-1} \vee \neg P_{n-1},..... \Rightarrow P_1 \vee \neg P_1 \rightarrow S$
.....
$P_n \vee \neg P_n , \Rightarrow (P_{n-1} \vee \neg P_{n-1}).....\rightarrow (P_1 \vee \neg P_1 \rightarrow S)$

$\vdash (P_n \vee \neg P_n) \rightarrow ((\ P_{n-1} \vee \neg P_{n-1}).....\rightarrow (P_1 \vee \neg P_1 \rightarrow S))..)$

Para obtener *S* hacemos la siguiente deducción axiomática:

1	$\vdash (P_n \vee \neg P_n) \rightarrow ((\ P_{n-1} \vee \neg P_{n-1}) \ldots\ldots \rightarrow (\ P_1 \vee \neg\ P_1 \rightarrow S))\ldots)$	
2	$P_n \vee \neg P_n$	
3	$\vdash ((\ P_{n-1} \vee \neg P_{n-1}) \ldots\ldots \rightarrow (\ P_1 \vee \neg\ P_1 \rightarrow S))\ldots)$	MP 1, 2
.....		
n-2)	$P_1 \vee \neg\ P_1$	
n-1)	$\vdash P_1 \vee \neg\ P_1 \rightarrow S$	
n)	S	MP (n-1), (n-2)

c.q.d.

3.3. Decidibilidad

Un sistema formal es decidible si dispone de un procedimiento general y finalizable para la decisión de la validez de fórmulas, es decir, la decidibilidad comprueba la equivalencia entre los conceptos sintácticos y semánticos.

El procedimiento general y finalizable para decidir si una fórmula es demostrable en un sistema, consiste en evaluar su tabla de verdad; si resulta ser semánticamente válida, entonces también será formalmente válida, y en caso contrario no. En la lógica proposicional, los sistemas formales son por tanto decidibles.

4. METALÓGICA DEL CÁLCULO DE PREDICADOS

La metalógica en el cálculo de predicados demuestra que el sistema es consistente, completo pero no es decidible en su totalidad, es decir que es parcialmente decidible (semidecidible).

4.1. Consistencia

Al igual que en el cálculo de proposiciones, se trata de probar que si una fórmula es válida, entonces es semánticamente correcta. Demostraremos la tautología utilizando el método del contrajemplo.

Tenemos que demostrar dos cosas:

a) Los axiomas son fórmulas semánticamente válidas

b) Las reglas de inferencia son tautológicas y por lo tanto garantizan que las fórmulas que se obtienen mediante su aplicación también lo son.

a) Demostración de que los axiomas son semánticamente válidos.

A9: $\vdash \forall x P(x) \rightarrow P(a)$

Demostración: Suponemos que la fórmula es F, $\forall x\ P(x) = V$ y $P(a) = F$; sin embargo si existe algún individuo que hiciera falso P(x) también haría falso $\forall x P(x)$ por las propiedades semánticas de los cuantificadores #, luego $\models$A9.

A10: Análogo.

b) Demostración de que las reglas de inferencia son semánticamente válidas.

- Regla de generalización universal condicional.

$$\frac{\vdash A \rightarrow B(a) \quad (1)}{\vdash A \rightarrow \forall x B(x) \quad (2)}$$

Demostración: Vamos a demostrar que si es semánticamente válida (1) entonces es semánticamente válida (2), es decir, que si no hay contrajemplo para (1) no lo hay para (2). Suponemos que (1) es semánticamente válida y que (2) no lo es. Entonces (2) = F , existe una interpretación *I* en donde , *A* = V y $\forall x\ B(x) = F$, luego en *I* hay un individuo x_i para el que $B(x_i)$ es F, en este caso se puede definir una interpretación *I'* para $A \rightarrow B(a)$ en el mismo dominio que *I*, que asignará 'a' a x_i. Para esta interpretación $A \rightarrow B(a)$ sería falsa, lo que no puede ocurrir, ya que (1) es semánticamente válida, es decir que es verdadera, para toda interpretación en todo dominio. # luego es semánticamente válida (2).

- Regla de generalización existencial condicional.

$$\frac{\vdash P(a) \rightarrow B \quad (1)}{\vdash \exists x P(x) \rightarrow B \quad (2)}$$

Demostración: De manera similar a la regla anterior

Conclusiones de la consistencia

1ª Conclusión:

> *"Toda fórmula demostrable es semánticamente válida en el sistema, ya que cualquier demostración tiene su origen en uno o varios axiomas y aplicación reiterada de las reglas de inferencia que transmiten la validez semántica."*

Por lo tanto el sistema no es contradictorio, es decir no se puede demostrar una fórmula y su negación. Si es demostrable *A* cualquier asignación asigna el valor verdadero a la fórmula luego asignaría el valor F a $\neg A$, que no es demostrable por no ser válida.

2ª Conclusión:

> *Toda deducción formalmente correcta es semánticamente correcta.*

Si tenemos $P_1, P_2, \ldots P_n \Rightarrow Q$, de lo que se trata es de probar que toda interpretación que verifica el conjunto de premisas $P_1, P_2, \ldots P_n$ asigna también significado verdadero a Q.

Demostración:

Q es una fórmula final de una secuencia deducida $R_1, R_2, \ldots R_m$ donde:

a) R_i es o una premisa o un axioma o un teorema.
b) R_i se obtiene aplicando alguna regla de inferencia.

Por el concepto semántico de deducción, hay que demostrar que toda interpretación *I* que satisface a las premisas, satisface también a los elementos de la secuencia R_i .

a) Nos presenta los siguientes casos de R_i :

> a.1.- Si es una premisa, es verdadera siempre, por la hipótesis de que *I* satisface todas las premisas.
>
> a.2.- Si es axioma o teorema, de acuerdo con lo demostrado anteriormente, por ser demostrable es semánticamente válida, y por tanto para cualquier interpretación, por ejemplo *I*, su significado es verdadero.

b) Nos presenta los siguientes casos de R_i :

b.1.- Si R_i se obtiene por MP, como se demostró esta regla obtiene fórmulas verdaderas a partir de premisas verdaderas, luego R_i será verdadera para *I*.

b.2.- Si R_i se obtiene por aplicación de generalización universal condicional a una premisa, si *I* no satisface $R_i = A \rightarrow \forall xP(x)$, tenemos que en *I*, *A* es verdad y $\forall xP(x)$ es falso, entonces, en algún elemento del dominio, por ejemplo *a*, P(*a*) es falsa, luego $A \rightarrow P(a)$ no sería válida en *I* por lo que no se puede aplicar esta regla en la deducción, #.

b.3.- De forma similar con la regla de generalización existencial condicional.

Luego, cualquiera de los casos anteriores es aplicable a R_1, luego R_1 es satisfacible por *I*. Si R_1 es satisfacible, R_2 puede ser también uno de los casos anteriores, y si R_1 es satisfacible R_2 también, y si lo son R_1, R_2 también lo es R_3 , etc, así hasta $R_m = Q$.

Luego si hay alguna interpretación que satisface las premisas satisface también la conclusión.

c.q.d.

4.2. Completud

Para demostrar la completud del sistema tenemos que probar que toda fórmula semánticamente válida |= *A* es demostrable en el sistema |-*A*. La completud relaciona el concepto semántico de verdad lógica con el concepto sintáctico de deducibilidad.

si |= *A entonces* |- *A*

Tenemos que demostrar que en cálculo de predicados se dará la completud si todas las fórmulas que representan verdades lógicas son formalmente deducibles en el sistema.

La completud del cálculo de predicados fue probada por Gödel[1], su teorema estaba dirigido a demostrar que si toda fórmula del cálculo de Predicados de primer orden es tautología, entonces es derivable en teoría de la demostración. Para ver este resultado, se va a emplear el procedimiento debido a Kleene y Beth, consistente en construir un sistema de deducción natural sobre un tipo de estructuras deductivas más general.

[1] K. Gödel, "Die Vollständigkeit der Axiome des logischen Funktionenkalkulus" (La completud de los axiomas del cálculo funcional lógico), Monatshefte für Mathematik und Phisyk, vol. 37, 1930

El teorema de Completud de Gödel.

> *"Para toda fórmula A de la lógica cuantificacional de primer orden, si A es lógicamente verdadera, entonces A es deducible"*
>
> si |= A entonces |- A

Demostración:

- Se parte de que *A* es lógicamente verdadera |=*A* ,entonces ¬*A* es insatisfacible (por definición de fórmula lógicamente verdadera).

- Si ¬*A* es insatisfacible, entonces ¬*A* es inconsistente (contraposición del Teorema de Henkin: toda fórmula consistente es satisfacible).

- Si ¬*A* es inconsistente entonces tenemos contradicción de la forma:
  ¬ A |- B y ¬A |- ¬B

Si ¬A |- B entonces por T. Deducción |- ¬A → B
Si ¬A |- ¬B entonces por T. Deducción |- ¬A → ¬B

Luego,

- 1	|- ¬A → B	
- 2	|- ¬A → ¬B	
3	|- (¬A → B) →((¬A → ¬B) → ¬¬A)	Reducción al absurdo
4	|- (¬A → ¬B) → ¬¬A	MP 1,3
5	|- ¬¬A	MP 2,4
6	|- ¬¬A → A	Ax.8
7	|- A	MP 5,6

c.q.d.

Procedimiento de Kleene y Beth.

Presentación

El procedimiento de Kleene, que está basado en la ideas de Beth, construye un sistema de deducción natural sobre un tipo de estructuras deductivas, de las cuales son un caso particular las que ya conocemos. La idea del procedimiento está en demostrar que si este nuevo tipo de estructuras deductivas son semánticamente correctas entonces son demostrables y que si ellas lo son, al ser un caso particular las vistas en LPO, éstas también cumplirán la misma propiedad.

Para trabajar con este nuevo tipo de estructuras tenemos que definir un sistemas de reglas, y a partir de ellas, o bien se consigue formalizar el proceso de demostración cuando la estructura es correcta, o bien se consigue un contraejemplo cuando la estructura no es correcta.

SISTEMA DE REGLAS DE KLEENE Y BETH

(IA$\rightarrow$)	$\dfrac{\Gamma \Rightarrow A, \Delta \,,, B \,, \Gamma \Rightarrow \Delta}{A \rightarrow B \,, \Gamma \Rightarrow \Delta}$	(IC$\rightarrow$)	$\dfrac{\Gamma, A \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \rightarrow B}$
(IA$\wedge$)	$\dfrac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta}$	(IC$\wedge$)	$\dfrac{\Gamma \Rightarrow \Delta \,, A \,,, \Gamma \Rightarrow \Delta \,, B}{\Gamma \Rightarrow \Delta \,, A \wedge B}$
(IA$\vee$)	$\dfrac{\Gamma \Rightarrow \Delta, A}{A \vee B, \Gamma \Rightarrow \Delta}$	(IC$\vee$)	$\dfrac{\Gamma \Rightarrow \Delta, A \vee B}{\Gamma \Rightarrow \Delta, A \vee B}$
(IA$\neg$)	$\dfrac{\Gamma \Rightarrow \Delta, A}{\Gamma \,, \neg A \Rightarrow \Delta}$	(IC$\neg$)	$\dfrac{A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A}$
(IA$\exists$)	$\dfrac{A(a), \Gamma \Rightarrow \Delta}{\exists x\, A(x), \Gamma \Rightarrow \Delta}$	(IC$\exists$)	$\dfrac{\Gamma \Rightarrow \Delta, A(a), \exists x A(x)}{\Gamma \Rightarrow \Delta, \exists x\, A(x)}$
(IA$\forall$)	$\dfrac{A(a), \forall x A(x), \Gamma \Rightarrow \Delta}{\forall x\, A(x), \Gamma \Rightarrow \Delta}$	(IC$\forall$)	$\dfrac{\Gamma \Rightarrow \Delta, A(a)}{\Gamma \Rightarrow \Delta, \forall x\, A(x)}$

Nota: Las reglas (IC$\forall$) y (IA$\exists$) tienen la condición de que la variable libre generalizada, no aparezca libre en el resto de las fórmulas de la secuencia, tanto en el antecedente como en el consecuente.

Reglas estructurales utilizadas en las demostraciones:

Atenuación:	$\underline{A \Rightarrow B}$ $A, C \Rightarrow B$	Permutación:	$\underline{A,B,C \Rightarrow D}$ $B,C,A \Rightarrow D$
Contracción:	$\underline{A, B, B \Rightarrow D}$ $A,B \Rightarrow D$	Corte:	$\underline{\Gamma \Rightarrow B,, \Delta,B \Rightarrow C}$ $\Delta, \Gamma \Rightarrow C$
Inclusión:	$\underline{A,B,C \Rightarrow D}$ $A \wedge B \wedge C \Rightarrow D$		

Proceso para la demostración de una estructura deductiva:

1º.- Estructura deductiva a estudiar.

Si partimos de una fórmula válida, transformar a estructura deductiva mediante el recíproco del teorema de deducción.

2º.- Obtención de nuevas estructuras mediante aplicación de reglas del sistema.

2.A.- Aplicación sólo de reglas conectivas.

- Elección de reglas aplicadas a la conectiva principal de la fórmula.
- Reiterar el proceso, si es conveniente.
- Si la estructura tiene forma de axioma, parar.
- Si la estructura tiene distintas componentes en antecedente y consecuente, ya no se puede aplicar ninguna regla, construcción de contraejemplo finito.
- Si la estructura tiene cuantificación pasar al punto 2.B.

2.B.- Aplicación de reglas de cuantificación (IC$\forall$) y (IA$\exists$).

- Generar un conjunto de variables libres.
- Para aplicar (IC$\forall$) y (IA$\exists$), hay que considerar que la variable a la que se aplica no esté libre en la estructura, si lo está, cambiar de nombre y añadirla al conjunto de variables libres.
- Reiterar el proceso, si es conveniente.
- Si la estructura es un axioma, parar.
- Si la estructura tiene distintas componentes en antecedente y consecuente, construcción del contraejemplo finito.
- Si la estructura tiene átomos cuantificados universalmente en el antecedente y existencialmente en el consecuente pasar al punto 2.C.

2.C.- Aplicación de reglas de cuantificación (IA∀) y (IC∃).

- Estas reglas no llevan restricciones de aplicación a las variables libres que aparecen en la fórmula.
- No se produce reducción de símbolos en las fórmulas.
- Aparecen sucesiones de instancias de las fórmulas cuantificadas.
- La prioridad que se establece al aplicar las reglas es aquella que a ser posible vaya consiguiendo fórmulas atómicas, tanto en antecedente como en cosecuente.
- Utilizar en cada aplicación para cada una de las fórmulas, la lista de variables ya existentes.
- Si se terminan las variables libres y no se tiene la estructura de axioma, se inicia un proceso de generación de nuevas variables, aplicando de nuevo 2.B.
- Si la estructura tiene forma de axioma, parar.
- Si aplicamos 2.B y 2.C indefinidamente, método del contraejemplo infinito.

3º.-Aplicación reiterada de 2 hasta obtener estructuras constituidas por fórmulas atómicas.

Método del contraejemplo:

* Contraejemplo finito:

Se aplica a estructuras que tienen distintos elementos en antecedente y consecuente , es decir, de la forma:

$$A_1, A_2, \ldots\ldots A_m \Rightarrow B_1, B_2, \ldots B_n$$

donde :

- todos los A_i son distintos de los B_i
- A_i y B_i son fórmulas atómicas cuantificadas o no y si se refieren al mismo predicado, tienen en sus argumentos distintas variables.
- el conjunto de variables libres es llamado V_i.

Proceso:

- Se establece una relación con los N tal que a cada variable de V_i se le asigna el número de orden que ocupa.

- Los predicados que aparecen en la secuencia se interpretan asi:
  Si $P(V_i)$ aparece en A_i , $P(i) = V$
  Si $P(V_i)$ no aparece en A_i , $P(i)= F$
  --- No puede aparecer $P(V_i)$ en ámbas pues entonces sería un axioma. ---

Por construcción esta interpretación es un contraejemplo.

4.3. Decisión

El problema de la decisión como ya sabemos trata de demostrar que existe un procedimiento mecánico o algoritmo que permite determinar en una serie de pasos finitos, si una fórmula *A* es o no deducible en el sistema (es decir si *A* es o no teorema).

En cálculo de predicados esta propiedad no es tan sencilla de demostrar como en cálculo de proposiciones. En 1936 Church[2] demostró que no existe un procedimiento efectivo que permita resolver el problema de la decisión en la lógica cuantificacional, ya que según él, dicha lógica es indecidible; sin embargo hay algunos aspectos de ella que resultan decidibles, por lo que se habla de semidecidibilidad.

5. TRABAJOS COMPLEMENTARIOS

Estudio introductorio a las lógicas no monótonas.

6. LA LÓGICA EN LA VIDA

1. *Metalenguaje* :

Hay una distinción implícita muy importante en lógica entre el lenguaje del objeto y el metalenguaje (noción ideada y desarrollada por el matemático polaco Alfred Tarski). Los enunciados en el nivel del objeto son enunciados dentro de un sistema formal. Por ejemplo:

$p \wedge q \rightarrow \neg r$
$\forall x \exists y P(x, y)$
«La nieve es blanca»

Los enunciados de metanivel son enunciados sobre el sistema o sobre los enunciados del nivel del objeto dentro de dicho sistema formal. Así tenemos, por ejemplo:

El enunciado *P* es verdadero
El argumento no es válido
El enunciado «La nieve es blanca» es verdadero

[2] Alonzo Church, "An unsolvable problem of elementary number theory", American Journal of Mathematics, vol. 58, 1936.
Alonzo Church, "A note on the Entscheidungsproblem",Journal of Simbolic Logic, vol I, 1936.

Así, si estamos estudiando la gramática inglesa, el inglés es la lengua-objeto y el castellano es la metalengua.

Para hablar de valores de verdad para enunciados de un metalenguaje utilizamos un metalenguaje de nivel superior. Así, cada peldaño de esta escalera infinita es metalenguaje del peldaño inmediatamente inferior, y a su vez, lenguaje objeto del peldaño situado sobre él.

Esta oración tiene tres errores

2. *Metadeseo* :

"Genio: En señal de gratitud por su heroica acción, me gustaría ofrecerle, de parte de mi Lámpara, la oportunidad de ver realizados tres de sus deseos.

Aquiles: ¡Qué asombroso! ¿No lo cree así, Sr. T?

Tortuga: Seguro que sí. Adelante, Aquiles, pida el primer deseo.

Aquiles: ¡Guau! ¿Pero qué podría desear? Oh, ¡ya sé! Es lo que pensé la primera vez que leí LAS MIL Y UNA NOCHES (esa colección de cuentos ingenuos (e incrustados uno dentro del otro)) - ¡desearía que pudiera pedir CIEN deseos en lugar de sólo tres! Bastante astuto, eh, ¿Sr. T? Siempre me he preguntado por qué aquellos lerdos personajes de las historias nunca lo intentaron.

Tortuga: Quizás ahora descrubrirá la respuesta.

Genio: Lo siento, Aquiles, pero no concedo metadeseos.

Aquiles: ¡Desearía que me contara qué es un "metadeseo"!

Genio: Pero ESO es un meta-metadeseo, Aquiles, y tampoco los concedo.

Aquiles: ¿Queé? No le entiendo nada.

Tortuga: ¿Por qué no repite su última petición, Aquiles?

Aquiles: ¿Qué quiere decir? ¿Por qué debería hacerlo?

Tortuga: Bueno, Ud. comenzó diciendo "yo deso". Ya que sólo está pidiendo información, ¿por qué no hace simplemente una pregunta?

Aquiles: Está bien, aunque no veo por qué. Digame, Sr. Genio, ¿qué es un metadeseo?

Genio: Simplemente es un deseo acerca de deseos. No se me permite conceder metadeseos. Sólo está a mi alcance conceder deseos comunes y corrientes, tales como desear tener diez botellas de cerveza, tener a Helena de Troya entre las sábanas, o tener un fin de semana para dos personas con todos los gastos pagados en Copacabana. Ud. sabe, cosas simples como ésas. Pero metadeseos no puedo conceder."

Para intentar conceder el metadeseo, el Genio saca de su manto una *Meta-Lampara*, la frota y entre una inmensa humareda aparece un *Meta-Genio*. Pero, quien desee saber más que se lea el libro.

Douglas Hofstadter
"*Gödel, Escher, Bach: un Eterno y Grácil Bucle*"

3. *Metaproblema* :

"Los *metaproblemas* son, por así decirlo, problemas acerca de problemas. Uno resuelve un metaproblema en base al conocimiento de que otro problema, o problemas, puede o no ser resuelto. ¡Estos problemas pueden ser bastante intrincados!"

"Un lógico de nuestro planeta visitó Og y se encontro con un nativo en una noche oscura y le preguntó si era un norteño verde. El nativo respondió (sí o no), pero el lógico no podía decir, a partir de su respuesta, de qué clase era.

Un segundo lógico se encontró con el mismo nativo en otra noche oscura y le preguntó si era un sureño verde. El nativo respondió (sí o no), pero el lógico no pudo descubrir qué era.

En otra noche oscura, un tercer lógico se encontró con el nativo y le preguntó si era un sureño rojo. El nativo respondió (sí o no), pero el lógico no pudo descubrir qué era.

¿De qué clase era?"

"Algunos problemas están en un nivel más profundo y pueden resolverse ¡sólo si se sabe si ciertos metaproblemas pueden ser resueltos! Mi tio llamaba a estos problemas *metametaproblemas*."

"El Brujo pensó por un momento. Bueno, dijo, mi tío, cierta vez, me planteó un metaproblema acerca de un lógico que visitó el planeta Og y se encontró con un nativo en una noche oscura y estaba curioso por saber si el nativo era o no sincero. Entonces, le preguntó al nativo de qué clase era (norteño verde, norteño rojo, sureño verde, sureño rojo) y el nativo nombró una de las cuatro clases y afirmo ser de esa clase.

Oh, dijo Annabelle, ¿pudo, entonces, el lógico determinar si el nativo era sincero o no?

Buena pregunta, contestó el Brujo, y justamente eso le pregunté a mi tío.

¿Le contestó?, preguntó Alexander.

Sí.

¿Qué respuesta le dio?, preguntó Alexander.

Eso no se lo diré.

Bueno, entonces, dijo Annabelle, después de que su tío le dijo si el lógico pudo o no resolver su problema, ¿pudo usted determinar si el nativo era sincero o no?

Tampoco se los diré, contestó el Brujo. Si les dijese eso, entonces ustedes podrían determinar si el nativo era o no sincero.

Entonces, ¿por qué no quiere decírnoslo?, preguntó Alexander. ¿No quiere que nosotros resolvamos el problema?

Ya no es necesario, dijo el Brujo con una sonrisa. Tienen suficiente información para determinar si el nativo era sincero o no."

Raymond Smullyan
"*Satán, Cantor y el infinito*"

NOTA: El planeta de Og está habitado por dos razas: la gente verde y la gente roja. Además según en la zona que vivan (hemisferio norte o hemisferio sur) los habitantes son muy diferentes, de forma que los norteños verdes siempre dicen la verdad mientras que los norteños rojos simpre mienten; complementariamente, los sureños verdes mienten mientras que los sureños rojos dicen la verdad.

Capítulo **7**

NORMALIZACIÓN DE FÓRMULAS

En capítulos anteriores hemos visto como construir fórmulas bien formadas, y como obtener nuevas fórmulas a partir de otras ya conocidas. Pero, si nos fijamos, vemos que un mismo enunciado puede estar representado por distintas fórmulas. En este tema veremos como normalizar fórmulas bien formadas del Cálculo de Proposiciones, y de manera más general, del Cálculo de Predicados. Se trata de, por simple manipulación sintactica, obtener una única representación, que llamaremos Forma Normal. Estas formas normales son muy importantes cuando apliacmos la lógica a la informática. Además, mediante el estudio de estas formas normales, nos será más fácil la obtención del valor de verdad de dichas fórmulas. Así, veremos unos métodos puramente mecánicos que nos permitirán determinar el valor de verdad de una fórmula proposicional.

Veamos como podemos transformar fórmulas lógicas en otras equivalentes que nos permitirán trabajar mejor con ellas. Estas nuevas fórmulas, que llamaremos en general *Formas Normales*, son de gran importancia a la hora de aplicar la lógica a la informática, como veremos en los próximos capítulos. Esto que ya es interesante en fórmulas proposicionales (Forma Normal Conjuntiva y Forma Normal Disyuntiva), lo es aún más si cabe, en fórmulas lógicas donde pueden aparecer combinados cuantificadores y conectivas. En este caso es conveniente pasar todos los cuantificadores al principio, en la llamada Forma Normal de Prenex; incluso eliminar los cuantificadores existenciales, obteniendo la Forma Normal de Skolem. Y, avanzando en la normalización, llegaremos a la Forma Clausal, que nos permitirá conectar directamente con los conceptos pertinentes para nuestra breve incursión en el mundo de la Programación Lógica.

1. FORMAS NORMALES DEL CÁLCULO DE PROPOSICIONES

Cualquier fórmula del Cálculo de Enunciados la podemos, mediante una serie finita de transformaciones simbólicas, representar en **Forma Normal**. Una vez obtenida la forma normal correspondiente, es sencillo decidir si la fórmula original es tautología, contradicción o indeterminación. Existen dos tipos distintos de formas normales, a las que podemos reducir una misma fórmula proposicional: **Forma Normal Disyuntiva** (FND) y **Forma Normal Conjuntiva** (FNC). Una Forma Normal, en general, se caracteriza por:

- sólo contiene tres clases de conectivas: conjunción ($\wedge$), disyunción ($\vee$) y negación ($\neg$)
- el negador, si aparece, lo hará directamente adosado a fórmulas atómicas
- en particular:
  - si es FNC: toda conjunción aparece fuera de paréntesis y toda disyunción dentro
  - si es FND: toda disyunción aparece fuera de paréntesis y toda conjunción dentro.

- FORMA NORMAL CONJUNTIVA:

Una fórmula de lógica de enunciados está en **FNC** si y sólo si esa fórmula no contiene coimplicadores ni implicadores, y consiste en una conjunción de disyunciones

$$D_1 \wedge D_2 \wedge \ldots \wedge D_n \qquad (n \geq 1)$$

en las cuales (D_i) el disyuntor vincula solamente fórmulas atómicas o negaciones de fórmulas atómicas

$$p'_1 \vee p'_2 \vee \ldots \vee p'_m$$

donde cada p'_i es una fórmula atómica afirmada o negada (**literal**).

Ejemplos :

Las siguientes fórmulas proposicionales no están en FNC:

$p \rightarrow q$

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

$\neg(p \vee q) \wedge \neg s$

Las siguientes fórmulas proposicionales están en FNC:

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

$(p \vee q) \wedge \neg s$

Esta definición también contempla los casos límite, de manera que una disyunción única se puede considerar como una conjunción degenerada, es decir, con un sólo miembro (n=1); análogamente, una única conjunción se puede considerar como una conjunción de disyunciones degeneradas (disyunciones de un sólo miembro, m=1). Así, las siguientes fórmulas proposicionales están en FNC:

$p \vee q$

$p \wedge q \wedge \neg r$

- FORMA NORMAL DISYUNTIVA:

Una fórmula de lógica de enunciados está en **FND** si y sólo si esa fórmula no contiene coimplicadores ni implicadores, y consiste en una disyunción de conjunciones

$$C_1 \vee C_2 \vee \ldots \vee C_n \qquad (n \geq 1)$$

en las cuales el conjuntor (C_i) vincula solamente fórmulas atómicas o negaciones de fórmulas atómicas

$$p'_1 \wedge p'_2 \wedge \ldots \wedge p'_m$$

Ejemplos:

Las siguientes fórmulas proposicionales no están en FND:

$(p \rightarrow q) \vee q$

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

En cambio, si que están en FND las siguientes:

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

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

Similarmente a las FNC, podemos considerar los casos límite de FND como las disyunciones degeneradas (formadas por un sólo miembro) y las disyunciones de conjunciones elementales degeneradas:

$p \vee \neg q \vee r$

$p \wedge q \wedge r$

Método de reducción a Forma Normal:

Para transformar una fórmula del Cálculo Proposicional a Forma Normal, podemos seguir el siguiente método, que consta de 4 pasos mecánicos:

1º) REDUCCIÓN DE CONSTANTES LÓGICAS: consiste en la eliminación de los implicadores y coimplicadores que puedan aparecer en la fórmula original. Para ello podemos utilizar la definición del coimplicador en términos de implicación y conjunción, para eliminar los coimplicadores; y la definición del implicador en función del negador y la conjunción o del negador y la disyunción:

$$A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$$
$$A \rightarrow B \equiv \neg A \vee B$$
$$A \rightarrow B \equiv \neg(A \wedge \neg B)$$

2º) NORMALIZACIÓN DEL NEGADOR: interiorización de los negadores de manera que cada negador quede directamente adosado a una fórmula atómica. Para ello podemos utilizar las leyes de De Morgan y la Doble Negación:

$$\neg(A \vee B) \equiv (\neg A \wedge \neg B)$$
$$\neg(A \wedge B) \equiv (\neg A \vee \neg B)$$
$$\neg\neg A \equiv A$$

3º) EXTERIORIZACIÓN DE CONJUNTORES o DISYUNTORES: este tercer paso tendrá diferente tratamiento según queramos obtener FNC o FND. De cualquier manera, utilizaremos las leyes Distributivas.

a) FNC: sacar el conjuntor fuera del paréntesis, utilizando la propiedad Distributiva de la Disyunción respecto a la Conjunción:

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$
$$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$$

b) FND: sacar el disyuntor fuera del paréntesis. Para ello podemos utilizar la ley Distributiva de la Conjunción respecto de la Disyunción:

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$$
$$(A \vee B) \wedge C \equiv (A \wedge C) \vee (B \wedge C)$$

4º) SIMPLIFICACIÓN y ORDENACIÓN DE RESULTADOS: este último paso no es obligado, pero presentará la fórmula de una forma más adecuada. Para ello realizaremos las siguientes acciones:

4.1.) Ordenar alfabéticamente las disyunciones o conjunciones elementales. Esto lo podemos hacer gracias a la ley Conmutativa de la Disyunción y de la Conjunción, respectivamente:

$$A \vee B \equiv B \vee A$$
$$A \wedge B \equiv B \wedge A$$

4.2.) Suprimir redundancias, para lo que podemos utilizar las leyes de idempotencia y absorción:

$$A \vee A \equiv A$$
$$A \wedge A \equiv A$$
$$A \wedge (A \vee B) \equiv A$$
$$A \vee (A \wedge B) \equiv A$$

Ejemplos:

1.- *Obtener la **FNC** de la siguiente fórmula proposicional*:

$$\mathbf{(p \rightarrow (q \rightarrow r)) \rightarrow (p \wedge q \rightarrow r)}$$

1º) Reducción de constantes lógicas:

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

2º) Normalización del negador:

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

3º) Exteriorización del conjuntor:

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

4º) Simplificación y ordenación del resultado:

$$\mathbf{(p \vee \neg p \vee \neg q \vee r) \wedge (\neg p \vee q \vee \neg q \vee r) \wedge (\neg p \vee \neg q \vee r \vee \neg r)}$$

2.- *Obtener la **FND** de la siguiente fórmula proposicional:*

$$\mathbf{p \wedge q \wedge \neg r \leftrightarrow \neg p \vee \neg q \vee r}$$

1º) Reducción de constantes lógicas:

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

2º) Normalización del negador:

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

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

3º) Exteriorización del disyuntor:

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

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

4º) Ordenación y simplificación de los resultados:

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

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

$$\mathbf{(p \wedge \neg p \wedge q \wedge \neg r) \vee (p \wedge q \wedge \neg q \wedge \neg r) \vee (p \wedge q \wedge r \wedge \neg r)}$$

3.- *Para cuando utilicemos la **propiedad distributiva** en la aplicación del paso 3, la podemos generalizar para cuando tenga varias componentes:*

$$\mathbf{(A \vee B) \wedge (C \vee D)}$$

$$[(A \vee B) \wedge C] \vee [(A \vee B) \wedge D]$$
$$[(A \wedge C) \vee (B \wedge C)] \vee [(A \wedge D) \vee (B \wedge D)]$$
$$\mathbf{(A \wedge C) \vee (B \wedge C) \vee (A \wedge D) \vee (B \wedge D)}$$

Podemos establecer que para cualquier fórmula del Cálculo de Proposiciones existe una Forma Normal Conjuntiva y una Forma Normal Disyuntiva equivalentes a ella. Esto nos permite disponer de un método mecánico para decidir si dicha fórmula es contradicción, tautología o indeterminación, ya que:

- Una FNC es una conjunción de disyunciones elementales. Si todas las disyunciones elementales son tautología, podemos decir que la fórmula será tautología. Para que una disyunción elemental sea tautología, en ella debe aparecer una fórmula atómica afirmada y negada a la vez. Resumiendo, podemos decir que *una FNC es TAUTOLOGÍA si en cada una de las disyunciones elementales aparece una fórmula atómica afirmada y negada*.

- Una FND es una disyunción de conjunciones elementales. Si todas las conjunciones elementales son contradicción, podemos decir que la fórmula será contradicción. Para que una conjunción elemental sea contradicción, en ella debe

aparecer una fórmula atómica afirmada y negada a la vez. Con lo que podemos decir que *una FND es CONTRADICCIÓN si en cada una de las conjunciones elementales aparece una fórmula atómica afirmada y negada*.

- Si una fórmula no es tautología ni contradicción, será *INDETERMINACIÓ*N.

Ejemplos:

La fórmula proposicional del ejemplo 1 es TAUTOLOGÍA, ya que una vez obtenida la FNC vemos que en cada disyunción elemental aparece una misma variable proposicional afirmada y negada. De igual forma, la fórmula proposicional del ejemplo 2 es una CONTRADICCIÓN, porque en cada conjunción elemental de la FND contiene alguna variable proposicional afirmada y negada al mismo tiempo.

3.- Determinar mediante Formas Normales el valor de verdad de la siguiente fórmula proposicional:

$$\neg(p \leftrightarrow q)$$

a) Reducimos a FND:

1º) Reducir constantes lógicas:

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

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

2º) Normalizar negador:

$$\neg(\neg p \vee q) \vee \neg(\neg q \vee p)$$

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

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

3º) Exteriorizar disyunciones:

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

4º) Ordenar y simplificar:

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

Podemos decir que no es contradicción ya que no aparece en todas las conjunciones una misma variable proposicional afirmada y negada a la vez.

b) Reducimos a FNC: los dos primeros pasos son iguales.

3º) Exteriorizar conjunciones:

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

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

4º) Ordenar y simplificar:

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

Podemos decir que no es tautología ya que no aparece en todas las disyunciones una misma variable proposicional afirmada y negada a la vez.

Por tanto podemos afirmar que esta fórmula proposicional es INDETERMINACIÓN.

4.- Determinar mediante Formas Normales el valor de verdad de la siguiente fórmula:

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

a) Reducimos a FND:

1º) Reducir constantes lógicas:

$$\neg(\neg p \vee (\neg q \vee p))$$

2º) Normalizar negador:

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

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

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

3º) Exteriorizar disyuntores:

$$p \wedge q \wedge \neg p$$

4º) Ordenar y simplificar:

$$p \wedge \neg p \wedge q$$

Esta formada por una disyunción degenerada (un sólo miembro). Como en cada conjunción elemental, en este caso sólo existe una, hay una proposición atómica afirmada y negada (*p* y ¬*p*) podemos afirmar que es CONTRADICCIÓN.

5.- Determinar mediante Formas Normales el valor de verdad de la siguiente fórmula:

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

a) Reducir a FNC:

1º) Reducción de constantes lógicas:

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

2º) Normalización del negador:

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

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

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

3º) Exteriorización del conjuntor:

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

4º) Simplificación y ordenación del resultado:

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

Como cada disyunción elemental tiene una proposición atómica afirmada y negada (*p* en la primera, *q* en la segunda y *r* en la tercera) la fórmula será TAUTOLOGÍA.

2. FORMAS NORMALES DEL CÁLCULO DE PREDICADOS

A partir de ahora y en adelante, los conceptos que veamos para desarrollar nuestra teoría, fundamentalmente los relacionados con la automatización, usarán las expresiones lógicas escritas en forma clausal, ya que esta notación tiene ventajas a la hora de manipular las fórmulas lógicas para su tratamiento informático. Toda fbf del Cálculo de Predicados tiene su *Forma Clausal* equivalente. Vamos a definir un procedimiento que realice la transformación de las estructuras lógicas a otras que sean equivalentes a ellas y que estén expresadas como disyunción de literales; esta representación, será la que se llame Forma Clausal (FC). Previamente veremos dos clases de formas normales para fórmulas del Cálculo de Predicados, la *Forma Normal de Prenex* y la *Forma Normal de Skolem*, sobre las que se apoya la transformación a forma clausal.

- FORMA NORMAL DE PRENEX

Como su nombre indica, una forma normal de prenex es la que tiene todos los cuantificadores en la cabeza de la fórmula y todas las conectivas detrás. Hay varias razones para interesarnos este tipo de representación. Una de ellas es que nos da una medida de la complejidad lógica de la sentencia asociada a la fórmula. Y esto no viene determinado por el número de cuantificadores sino por el número de alternancias entre $\forall$ y $\exists$. La otra es que esta forma es similar a las formas normales proposicionales (libres de cuantificadores).

Decimos que una fbf está en **FNP** si tiene las siguientes características:

- Todos los cuantificadores aparecen en cabeza de una expresión a la que afectan en su totalidad.

- La expresión que está afectada por estos cuantificadores se llama *matriz de la fórmula*, la cual está constituida por predicados y conectivas (solamente disyunción, conjunción y negación, ésta última sólo afectando a fórmulas atómicas).

Proceso para reducir una fbf a FNP:

1- Eliminar implicadores y coimplicadores: utilizando las reglas de equivalencia vistas para formas normales proposicionales.

2- Normalizar el negador: por medio de las leyes de De Morgan, Eliminación del Negador y las reglas de Negación de Cuantificadores.

3- En este punto tenemos que toda fórmula parcial está conectada a una cuantificada por disyunción ó conjunción. Tenemos dos casos:

a) La fórmula parcial no contiene libre la variable cuantificada, entonces se aplican las siguientes reglas:

$$A \vee \forall x\, P(x) \leftrightarrow \forall x\, (P(x) \vee A)$$
$$A \vee \exists x\, P(x) \leftrightarrow \exists x\, (P(x) \vee A)$$
$$A \wedge \forall x\, P(x) \leftrightarrow \forall x\, (P(x) \wedge A)$$
$$A \wedge \exists x\, P(x) \leftrightarrow \exists x\, (P(x) \wedge A)$$

b) La fórmula parcial sí contiene libre la variable cuantificada, entonces debemos aplicar:

$$A(x) \vee \forall x\, P(x) \leftrightarrow \forall y\, (A(x) \vee P(y))$$
$$A(x) \vee \exists x\, P(x) \leftrightarrow \exists y\, (A(x) \vee P(y))$$
$$A(x) \wedge \forall x\, P(x) \leftrightarrow \forall y\, (A(x) \wedge P(y))$$
$$A(x) \wedge \exists x\, P(x) \leftrightarrow \exists y\, (A(x) \wedge P(y))$$

Ejemplo:

Encontrar la **FNP** equivalente a la siguiente fbf:

$$A = \neg\exists x\, (P(x) \rightarrow \forall y\, Q(y))$$

1	$\neg\exists x\, (\neg P(x) \vee \forall y\, Q(y))$	I, DI A
2	$\forall x\, \neg(\neg P(x) \vee \forall y\, Q(y))$	NE 1
3	$\forall x\, (\neg\neg P(x) \wedge \neg\forall y\, Q(y))$	I, DM 2
4	$\forall x\, (P(x) \wedge \neg\forall y\, Q(y))$	I, EN 3
5	$\forall x\, (P(x) \wedge \exists y\, \neg Q(y))$	I, NU 4
6	$\forall x\, \exists y\, (P(x) \wedge \neg Q(y))$	Aplicamos 3 a) porque y no aparece libre en P(x)

- FORMA NORMAL DE SKOLEM

Decimos que una fbf está en **FNS** si tiene las siguientes características:

- Todos los cuantificadores están en cabeza de la fórmula.

- Sólo existen cuantificadores universales.

- La matriz de la fórmula está constituida por una conjunción de fórmulas consistentes en una disyunción de literales.

Ya hemos visto como conseguir la primera y tercera características. Los pasos a seguir para pasar los cuantificadores a la cabeza de la fórmula están explicados en la forma normal de prenex. Y los pasos para obtener un conjunción de fórmulas consistentes en disyunciones de literales se ha explicado en la forma normal conjuntiva. Por tanto lo único que nos queda es ver como eliminar los cuantificadores existenciales. Para ello introduciremos dos conceptos nuevos: constante de Skolem y función de Skolem.

Proceso para introducir constantes y funciones de Skolem:

Debemos eliminar los cuantificadores existenciales para obtener la FNS. Esto se consigue por introducción de las constantes y funciones de Skolem, de acuerdo a las siguientes normas:

- Si el *cuantificador existencial está en el ámbito de un cuantificador universal,* entonces admitimos la posibilidad de que la variable del cuantificador existencial dependa del valor de la variable afectada por el cuantificador universal; esta dependencia la representamos por el nombre de una función. Por tanto, reemplazaremos todas las ocurrencias de esta variable cuantificada existencialmente por una ***función de Skolem*** cuyos argumentos serán aquellas variables cuantificadas universalmente en cuyo ámbito esté el cuantificador existencial que se elimina.

- Si el *cuantificador existencial a eliminar no está dentro del ámbito de ningún cuantificador universal*, sustituimos la variable cuantificada existencialmente por una función sin argumentos: ***constante de Skolem.***

Los símbolos de constantes (*a*, *b*, *c*, ...) y funciones de Skolem (*f*, *g*, *h*, ...) deben ser nuevos en la fórmula y distintos para cada cuantificador distinto eliminado. Podemos utilizar el símbolo *a* con subíndices para constantes y la letra *f* con subíndices para funciones.

Ejemplos:

1.- Poner en FNS las siguientes fbf:

$\exists x\ P(x)$	$P(a)$
$\exists x\ \exists y\ P(x,y)$	$P(a,b)$
$\exists x\ \forall y\ P(x,y)$	$\forall y\ P(a,y)$
$\forall x\ \exists y\ P(x,y)$	$\forall x\ P(x,f(x))$
$\forall x\ \forall y\ \exists z\ P(x,y,z)$	$\forall x\ \forall y\ P(x,y,f(x,y))$
$\forall x\ \exists y\ \exists z\ P(x,y,z)$	$\forall x\ P(x,f(x),g(x))$

2.- Poner en FNS la siguiente fbf:

$$\forall x \exists y \ (P(x, y) \wedge \neg Q(y))$$

Al estar el $\exists$ dentro del ámbito del $\forall$, la variable *y* es sustituida por una función de la variable cuantificada universalmente, es decir, *y* se sustituye por una función de Skolem: *f(x),* en todas las ocurrencias de *y* en la fórmula.

FNS: $\forall x \ (P(x, f(x)) \wedge \neg Q(f(x)))$

3.- Poner en FNS la siguiente fbf:

$$\exists x \forall y [P(x, y) \wedge \exists x \forall z \exists y (Q(y, z) \vee R(x, y))]$$

El primer existe está fuera de todo universal por lo que lo sustituiremos por una constante de skolem (*a*); el segundo existe, pese a tener como índice la variable *x*, no es la misma que la anterior, además de que está dentro del ámbito del universal con *y*, por tanto lo cambiaremos por *f(y)*; el tercer existe, está dentro del ámbito de dos universales, *y* y *z*, por lo que será sustituido por *g(y, z)*. Así la fórmula en **FNS** queda:

$$\forall y \{ P(a, y) \wedge \forall z [Q(g(y, z), z) \vee R(f(y), g(y, z))] \}$$

- FORMA CLAUSAL

Una forma clausal consiste en una colección de cláusulas; donde una cláusula, como ya hemos visto, está formada por una disyunción de literales; recordemos que un literal es una fórmula atómica afirmada o negada. Partiremos de una fórmula bien formada de la Lógica de Primer Orden que no tiene variables libres. Ahora ya estamos en condiciones de obtener la Forma Clausal de una fbf, y para ello seguiremos los pasos enumerados a continuación:

1º Eliminar implicadores y coimplicadores.

2º Normalizar negadores.

3º Normalizar variables: si es necesario, renombrar variables.

4º Eliminar cuantificadores existenciales: constantes y funciones de skolem.

5º Forma prenexa: adelantar los cuantificadores universales a la cabeza de la fórmula.

6º Eliminación de cuantificadores universales: llegados a este punto, sabemos que todas las variables que aparecen están cuantificadas universalmente, por lo que no los escribimos pero sabemos que están implícitamente.

7º Poner en FNC la matriz de la fórmula.

8º Extraer las cláusulas: eliminación de conjunciones.

9º Normalizar variables: si es necesario, volvemos a renombrar las variables.

Ejemplos:

1.- *Obtener la FC de la siguiente fbf:*

$A = \forall x \{ [P(x) \rightarrow \neg\forall y (Q(x, y) \rightarrow \exists z P(z))] \wedge \forall y (Q(x, y) \rightarrow R(y)) \}$

Solución:

1.- Eliminar implicadores y coimplicadores

$\forall x \{ [\neg P(x) \vee \neg\forall y (\neg Q(x, y) \vee \exists z P(z))] \wedge \forall y (\neg Q(x, y) \vee R(y)) \}$ I^3, DI

2.- Normalizar negador

$\forall x \{ [\neg P(x) \vee \exists y \neg(\neg Q(x, y) \vee \exists z P(z))] \wedge \forall y (\neg Q(x, y) \vee R(y)) \}$ I, NU
$\forall x \{ [\neg P(x) \vee \exists y (\neg\neg Q(x, y) \wedge \neg\exists z P(z))] \wedge \forall y (\neg Q(x, y) \vee R(y)) \}$ I, DM
$\forall x \{ [\neg P(x) \vee \exists y (\neg\neg Q(x, y) \wedge \forall z \neg P(z))] \wedge \forall y (\neg Q(x, y) \vee R(y)) \}$ I, NE
$\forall x \{ [\neg P(x) \vee \exists y (Q(x, y) \wedge \forall z \neg P(z))] \wedge \forall y (\neg Q(x, y) \vee R(y)) \}$ I, EN

3.- Normalizar variables

$\forall x \{ [\neg P(x) \vee \exists y (Q(x, y) \wedge \forall z \neg P(z))] \wedge \forall w (\neg Q(x, w) \vee R(w)) \}$

4.- Eliminar cuantificadores existenciales: forma de Skolem

$\forall x \{ [\neg P(x) \vee (Q(x, f(x)) \wedge \forall z \neg P(z))] \wedge \forall w (\neg Q(x, w) \vee R(w)) \}$

5.- Sacar cuantificadores universales fuera de la fórmula: forma Prenexa

$\forall x \{ [\neg P(x) \vee \forall z (Q(x, f(x)) \wedge \neg P(z))] \wedge \forall w (\neg Q(x, w) \vee R(w)) \}$
$\forall x \forall z \forall w \{ [\neg P(x) \vee (Q(x, f(x)) \wedge \neg P(z))] \wedge (\neg Q(x, w) \vee R(w)) \}$

6.- Eliminar cuantificadores universales

$[\neg P(x) \vee (Q(x, f(x)) \wedge \neg P(z))] \wedge (\neg Q(x, w) \vee R(w))$

7.- FNC de la matriz de la fórmula: aplicamos la propiedad distributiva

$[\neg P(x) \vee Q(x, f(x))] \wedge [\neg P(x) \vee \neg P(z)] \wedge [\neg Q(x, w) \vee R(w)]$

8.- Separamos las Cláusulas:

C_1: $\neg P(x) \vee Q(x, f(x))$
C_2: $\neg P(x) \vee \neg P(z)$
C_3: $\neg Q(x, w) \vee R(w)$

9.- Normalizamos variables

C_1: $\neg P(x_1) \vee Q(x_1, f(x_1))$
C_2: $\neg P(x_2) \vee \neg P(z)$

C_3: $\neg Q(x_3, w) \vee R(w)$

2.- *Obtener la forma clausal de la siguiente fórmula lógica:*

$$\forall x\, \exists y\, [\, \neg A(x,y) \vee (B(x) \wedge C(y))\,] \wedge \forall x\, [B(x) \rightarrow \neg \forall x\, \forall y\, D(x,y)] \wedge \neg \exists x\, E(x) \wedge [\, \forall x\, \forall y\, D(x,y) \vee \exists x\, E(x)\,]$$

1. Eliminar implicadores y coimplicadores:
$\forall x\, \exists y\, [\, \neg A(x,y) \vee (B(x) \wedge C(y))\,] \wedge \forall x\, [\neg B(x) \vee \neg \forall x\, \forall y\, D(x,y)] \wedge \neg \exists x\, E(x) \wedge [\, \forall x\, \forall y\, D(x,y) \vee \exists x\, E(x)\,]$

2. Normalizar negadores:
$\forall x\, \exists y\, [\, \neg A(x,y) \vee (B(x) \wedge C(y))\,] \wedge \forall x\, [\neg B(x) \vee \exists x\, \exists y\, \neg D(x,y)] \wedge \forall x\, \neg E(x) \wedge [\, \forall x\, \forall y\, D(x,y) \vee \exists x\, E(x)\,]$

3. Normalizar variables:
$\forall x\, \exists y\, [\, \neg A(x,y) \vee (B(x) \wedge C(y))\,] \wedge \forall z\, [\neg B(z) \vee \exists v\, \exists w\, \neg D(v,w)] \wedge \forall u\, \neg E(u) \wedge [\, \forall s\, \forall t\, D(s,t) \vee \exists r\, E(r)\,]$

4. Eliminar cuantificadores existenciales:
$\forall x\, [\, \neg A(x,f(x)) \vee (B(x) \wedge C(f(x))\,] \wedge \forall z\, [\, \neg B(z) \vee \neg D(g(z),h(z))\,] \wedge \forall u\, \neg E(u) \wedge [\, \forall s\, \forall t\, D(s,t) \vee E(a)\,]$

5. Forma prenexa:
$\forall x\, \forall z\, \forall u\, \forall s\, \forall t\, \{\, [\, \neg A(x,f(x)) \vee (B(x) \wedge C(f(x))\,] \wedge [\, \neg B(z) \vee \neg D(g(z), (z))\,] \wedge \neg E(u) \wedge [\, D(s,t) \vee E(a)\,]\, \}$

6. Eliminar cuantificadores universales:
$[\, \neg A(x, f(x)) \vee (B(x) \wedge C(f(x))\,] \wedge [\, \neg B(z) \vee \neg D(g(z), h(z))\,] \wedge \neg E(u) \wedge [\, D(s, t) \vee E(a)\,]$

7. FNC de la matriz de la fórmula:
$[\, \neg A(x, f(x)) \vee B(x)\,] \wedge [\, \neg A(x, f(x)) \vee C(f(x))\,] \wedge [\, \neg B(z) \vee \neg D(g(z), h(z))\,] \wedge \neg E(u) \wedge [\, D(s, t) \vee E(a)\,]$

8. Extracción de cláusulas: eliminación de conjunciones
C_1: $\neg A(x, f(x)) \vee B(x)$
C_2: $\neg A(x, f(x)) \vee C(f(x))$
C_3: $\neg B(z) \vee \neg D(g(z), h(z))$
C_4: $\neg E(u)$
C_5: $D(s, t) \vee E(a)$

9. Normalización de variables:
C_1: $\mathbf{\neg A(x_1, f(x_1)) \vee B(x_1)}$
C_2: $\mathbf{\neg A(x_2, f(x_2)) \vee C(f(x_2))}$
C_3: $\mathbf{\neg B(x_3) \vee \neg D(g(x_3), h(x_3))}$
C_4: $\mathbf{\neg E(x_4)}$
C_5: $\mathbf{D(x_5, x_6) \vee E(a)}$

3. MÉTODOS DE DEMOSTRACIÓN BASADOS EN FORMAS NORMALES

Son métodos aplicables a FBF escritas en Forma Normal y que nos sirven para establecer su valor de verdad: tautología, contradicción o indeterminación. Ya hemos estudiado una forma de obtener el valor de verdad de una fórmula a partir de la FNC y la FND. Ahora vamos a ver dos métodos distintos, cada uno aplicable a una de las

Formas Normales: el método del Cuadro para fórmulas en FND y el método de Davis-Putnam para fórmulas en FNC. Estos métodos se basan en las siguientes ideas generales:

- Para determinar si la fórmula es tautología, indeterminación o contingencia, no es necesario calcular el valor de verdad de todas las interpretaciones posibles (2^n siendo n el número de variables proposicionales distintas de la fórmula) tal como hacemos con las tablas de verdad. En algunos casos el valor de verdad de una determinada variable nos permite conocer el valor para un grupo de interpretaciones:

  Ejemplo:

  Sea la fbf

  $$p \wedge A$$

  siendo A una subfórmula bien formada cualquiera. Cuando p es falsa, el valor de la fórmula completa es:

  $$F \wedge A = F$$

  independientemente del valor que tomen las variables proposicionales que aparezcan en A. Por tanto, únicamente debemos analizar los casos en que p sea verdadera, por lo que nos ahorramos el estudio de la mitad de las interpretaciones.

  De forma análoga, sea la fbf

  $$p \vee A$$

  siendo A una subfórmula bien formada cualquiera. Cuando p es cierta, el valor de la fórmula completa es:

  $$V \vee A = V$$

  independientemente del valor que tomen las variables proposicionales que aparezcan en A. Por tanto, únicamente debemos analizar los casos en que p sea falsa, por lo que nos evitamos el estudio de la mitad de las interpretaciones.

- Cada paso del método disminuye en uno el número de variables proposicionales distintas de la fórmula, al estudiar lo que ocurre tanto cuando el valor de dicha variable es verdadero como cuando es falso. Por tanto, como el número de variables es finito, en un número finito de pasos el procedimiento finalizará, determinando la valoración semántica de la fórmula.

Recordemos, por último, la definición de **literal**: cualquier variable proposicional afirmada o negada.

3.1. MÉTODO DEL CUADRO

Este método es aplicable a Formas Normales Disyuntivas. Por tanto, de no estarlo, debemos pasar la fórmula a FND.

Los pasos a seguir son:

1) Si en todas las conjunciones elementales aparecen la afirmación y la negación de una misma fórmula atómica: CONTRADICCIÓN.
Si no están en todas, pero aparecen en algunas conjunciones, las eliminamos, ya que las hacen falsas y $F \vee p \equiv p$.

2) Si existen conjunciones elementales consistentes en un literal, la fórmula será cierta si lo es este literal. Por tanto, nos interesa determinar el valor de la fórmula cuando este literal tenga valor falso. Entonces se les asigna el valor de falso (y verdadero a su complementario) y reducimos la fórmula.
Este paso hay que repetirlo mientras tengamos alguna conjunción elemental consistente en un literal.

3) Escoger una conjunción elemental y descomponer la fórmula en:
$A = C \vee B = (l \wedge D) \vee B = (l \vee B) \wedge (D \vee B)$

donde l es un literal y aplicar el método a cada una de las partes:

3.1) $(l \vee B)$ ir al paso 2

3.2) $(D \vee B)$ ir a paso 2

Si obtenemos tautología, en todas y cada una de las partes: TAUTOLOGÍA
En otro caso: INDETERMINACIÓN

Ejemplo 1:

Determinar el valor de verdad de la siguiente fórmula proposicional, utilizando el Método del Cuadro:

$$\mathbf{(p \wedge q) \vee r \rightarrow \neg q \vee \neg r}$$

Como paso previo, debemos obtener la FND:

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

1) No es contradicción.

2) $\neg q \Rightarrow F$ $(\neg p \wedge \neg r) \vee (F \wedge \neg r) \vee F \vee \neg r$
$(\neg p \wedge \neg r) \vee F \vee F \vee \neg r$
$(\neg p \wedge \neg r) \vee \neg r$

$\neg r \Rightarrow F$ $(\neg p \wedge F) \vee F$

F ∨ F

F ⟹ No tautología

La fórmula es **CONTINGENCIA** o **INDETERMINACIÓN**

Ejemplo 2:

Determinar el valor de verdad de la siguiente fórmula proposicional, utilizando el Método del Cuadro:

p ∨ (¬p ∧ q ∧ r) ∨ (¬p ∧ ¬q ∧ r) ∨ (¬p ∧ ¬r) ∨ (p ∧ r)

Ya está en FND

1) No es contradicción

2) p ↯ F

F ∨ (V ∧ q ∧ r) ∨ (V ∧ ¬q ∧ r) ∨ (V ∧ ¬r) ∨ (F ∧ r)

F ∨ (q ∧ r) ∨ (¬q ∧ r) ∨ ¬r ∨ F

(q ∧ r) ∨ (¬q ∧ r) ∨ ¬r

¬r ↯ F

(q ∧ V) ∨ (¬q ∧ V) ∨ F

q ∨ ¬q ∨ F

q ∨ ¬q

V ⟹ **TAUTOLOGÍA**

3.2. MÉTODO DE DAVIS-PUTNAM

El procedimiento de Davis-Putnam[1] es un método clásico y eficiente para determinar la validez de una sentencia del cálculo proposicional. Este método es aplicable a fórmulas proposicionales que estén en Forma Normal Conjuntiva.

Los pasos a seguir son:

1) Si en cada una de las disyunciones elementales aparecen la afirmación y la negación de una misma fórmula atómica: TAUTOLOGÍA.
Si no aparecen en todas, pero sí en algunas, estas se eliminarán ya que hacen verdadera la cláusula y $V \wedge p \equiv p$.

[1] M. Davis y H. Putnam. "A computing procedure for quantification theory". *Journal of the ACM*, vol. 7 (3), pág. 201-215, 1960.

2) Si existen disyunciones elementales consistentes en un literal, si este fuese falso, toda la fórmula sería falsa; vamos a ver que ocurriría si tomara el valor verdadero. Entonces se les asigna el valor de verdadero (y falso a su complementario) y reducimos la fórmula.
Repetiremos este paso mientras queden disyunciones elementales consistentes en un sólo literal.

3) Si una variable proposicional p aparece únicamente en un estado, es decir, como p o como $\neg p$, se le asigna el valor de verdadero y reducimos la fórmula.

4) Llegados a este punto, todas las variables proposicionales aparecen afirmadas y negadas. Si aparece p y $\neg p$ descomponer la fórmula A en:

B: conjunción de las cláusulas (B_i) que contienen a p $\quad i=1..n$
C: conjunción de las cláusulas (C_j) que contienen $\neg p$ $\quad j=1..m$
D: cláusulas restantes (D_k) $\quad k=1..\tilde{n}$

y se reemplaza por A':

$$A' = [\bigwedge (B_i' \vee C_j')] \wedge D$$

donde B_i' y C_j' son las cláusulas B_i y C_j después de eliminar la variable proposicional p y $\neg p$, respectivamente. La nueva expresión A' tendrá por tanto (n x $m + \tilde{n}$) cláusulas.

Volver a aplicar el método a la nueva expresión A'.

Si obtenemos contradicción: CONTRADICCIÓN
En otro caso: INDETERMINACIÓN

Ejemplo 1:

Determinar el valor de verdad de la siguiente fórmula proposicional, utilizando el Método de Davis-Putnam:

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

Ya está en FNC

1) No es tautología

2) $\neg p \Rightarrow V$ $\quad V \wedge (F \vee q \vee r) \wedge (\neg q \vee r \vee s) \wedge (q \vee \neg r) \wedge (\neg q \vee \neg s)$
$(q \vee r) \wedge (\neg q \vee r \vee s) \wedge (q \vee \neg r) \wedge (\neg q \vee \neg s)$

3) No aparece ninguna variable proposicional solamente afirmada o solamente negada.

4) Descomponemos la fórmula, tomando q y $\neg q$:

B: ($\underline{g} \vee r) \wedge (\underline{g} \vee \neg r)$

C: ($\underline{\neg g} \vee r \vee s) \wedge (\underline{\neg g} \vee \neg s)$

D: $\varnothing$

quedándonos: $(r \vee r \vee s) \wedge (r \vee \neg s) \wedge (\neg r \vee r \vee s) \wedge (\neg r \vee \neg s)$

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

y aplicamos de nuevo el método:

2) No aparece ninguna disyunción elemental consistente en un sólo literal

3) No aparece ninguna variable proposicional solamente afirmada o negada

4) Separamos eligiendo la variable *r*:

B: ($\underline{r} \vee s) \wedge (\underline{r} \vee \neg s)$

C: ($\underline{\neg r} \vee \neg s)$

D: $\varnothing$

quedándonos: $(s \vee \neg s) \wedge (\neg s \vee \neg s)$

$V \wedge \neg s$

$\neg s \quad \Longrightarrow$ **INDETERMINACIÓN**

Ejemplo 2:

Determinar por el Método de Davis-Putnam el valor de verdad de fórmula proposicional:

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

Ya está en FNC

1) No es tautología

2) No existen disyunciones elementales consistentes en un sólo literal

3) No aparece ninguna variable proposicional solamente afirmada o solamente negada.

4) Descomponemos con *p*:

B: ($\underline{p} \vee q) \wedge (\underline{p} \vee \neg q)$

C: ($\underline{\neg p} \vee q) \wedge (\underline{\neg p} \vee \neg q)$

D: $(q \vee r) \wedge (q \vee \neg r)$

quedándonos: $(q \vee q) \wedge (q \vee \neg q) \wedge (\neg q \vee q) \wedge (\neg q \vee \neg q) \wedge (q \vee r) \wedge (q \vee \neg r)$

$q \wedge V \wedge V \wedge \neg q \wedge (q \vee r) \wedge (q \vee \neg r)$

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

y volvemos a aplicar el método:

2) $q \Rightarrow V \qquad V \wedge F \wedge (V \vee r) \wedge (V \vee \neg r)$

$$V \wedge F \wedge V \wedge V$$

$$F \qquad \Rightarrow \textbf{CONTRADICCIÓN}$$

4. EJERCICIOS

1.- *Obtener utilizando Formas Normales el valor de verdad de la siguiente fórmula proposicional:*

$$\mathbf{p \wedge \neg q \rightarrow q \vee \neg r}$$

Reducimos a Forma Normal:
1º) Reducir constantes lógicas:

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

2º) Normalizar negadores:

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

$$\neg p \vee q \vee q \vee \neg r$$

3º) Exteriorizar conjuntores o disyuntores:

$$\neg p \vee q \vee q \vee \neg r$$

4º) Ordenación y simplificación:

$$\neg p \vee q \vee \neg r$$

Está, al mismo tiempo, en FNC y FND. Como en cada disyunción elemental no aparece una variable proposicional y su negada no será tautología; ni en cada conjunción elemental aparece una proposición atómica y su negada, no será contradicción. Por tanto, será **INDETERMINACIÓN** o **CONTINGENCIA**.

2.- *Obtener utilizando Formas Normales el valor de verdad de la siguiente fórmula proposicional:*

$$\mathbf{((p \rightarrow q) \rightarrow r) \rightarrow q}$$

Reducimos a Forma Normal:
1º) Reducir constantes lógicas:

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

2º) Normalizar negadores:

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

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

3º) Exteriorizar conjuntores o disyuntores:

FNC

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

FND

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

4º) Ordenación y simplificación:

FNC

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

FND

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

Podemos ver que se trata de una **CONTINGENCIA**.

3.- *Obtener utilizando Formas Normales el valor de verdad de la siguiente fórmula proposicional:*

$$\mathbf{(p \wedge q) \vee r \rightarrow \neg q \vee \neg r}$$

Reducimos a Forma Normal:
1º) Reducir constantes lógicas:

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

2º) Normalizar negadores:

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

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

3º) Exteriorizar conjuntores o disyuntores:

FNC	FND
$(\neg p \vee \neg q \vee \neg q \vee \neg r) \wedge (\neg r \vee \neg q \vee \neg r)$	$((\neg p \wedge \neg r) \vee (\neg q \wedge \neg r)) \vee \neg q \vee \neg r$

4º) Ordenación y simplificación:

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

Como podemos observar la fórmula es una I**NDETERMINACIÓN**.

4.- *Obtener utilizando Formas Normales el valor de verdad de la siguiente fórmula proposicional:*

$$\mathbf{((p \rightarrow q) \rightarrow r) \rightarrow (p \rightarrow r) \wedge (q \rightarrow r)}$$

Reducimos a Forma Normal:

1º) Reducir constantes lógicas:

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

2º) Normalizar negadores:

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

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

3º) Exteriorizar conjuntores o disyuntores:

FNC	FND
$(((\neg p \vee q) \wedge \neg r) \vee (\neg p \vee r)) \wedge (((\neg p \vee q) \wedge \neg r) \vee (\neg q \vee r))$	$((\neg p \wedge \neg r) \vee (q \wedge \neg r)) \vee (((\neg p \vee r) \wedge \neg q) \vee ((\neg p \vee r) \wedge r))$
$((\neg p \vee q \vee \neg p \vee r) \wedge (\neg r \vee \neg p \vee r)) \wedge ((\neg p \vee q \vee \neg q \vee r) \wedge (\neg r \vee \neg q \vee r))$	$((\neg p \wedge \neg r) \vee (q \wedge \neg r)) \vee (((\neg p \wedge \neg q) \vee (r \wedge \neg q)) \vee ((\neg p \wedge r) \vee (r \wedge r)))$

4º) Ordenación y simplificación:

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

La fórmula es una **INDETERMINACIÓN**

5.- *Obtener la forma clausal de las siguientes fórmulas:*

a) $\forall x\, \forall y\, [\, sobre(x, y) \rightarrow encimaDe(x, y)\,]$

1. Eliminar implicadores y coimplicadores:
$\forall x\, \forall y\, [\, \neg sobre(x, y) \vee encimaDe(x, y)\,]$

2. Normalizar negadores:
3. Normalizar variables:
4. Eliminar cuantificadores existenciales:
5. Forma prenexa:
6. Eliminar cuantificadores universales:
$\neg sobre(x, y) \vee encimaDe(x, y)$

7. FNC de la matriz de la fórmula:
8. Extracción de cláusulas: eliminación de conjunciones
9. Normalización de variables:
C_1: $\neg sobre(x, y) \vee encimaDe(x, y)$

b) $\forall x\, \forall y\, \forall z\, [\, encimaDe(x, y) \wedge encimaDe(y, z) \rightarrow encimaDe(x, z)\,]$

1. Eliminar implicadores y coimplicadores:
$\forall x\, \forall y\, \forall z\, \{\, \neg[\, encimaDe(x, y) \wedge encimaDe(y, z)\,] \vee encimaDe(x, z)\,]\, \}$

2. Normalizar negadores:
$\forall x\, \forall y\, \forall z\, \{\, [\, \neg encimaDe(x, y) \vee \neg encimaDe(y, z)\,] \vee encimaDe(x, z)\,]\, \}$
3. Normalizar variables:
4. Eliminar cuantificadores existenciales:
5. Forma prenexa:
6. Eliminar cuantificadores universales:
$\neg encimaDe(x, y) \vee \neg encimaDe(y, z) \vee encimaDe(x, z)$

7. FNC de la matriz de la fórmula:
8. Extracción de cláusulas: eliminación de conjunciones
9. Normalización de variables:
C_1: $\neg encimaDe(x, y) \vee \neg encimaDe(y, z) \vee encimaDe(x, z)$

c) $\forall x\, \forall y\, \{\, encimaDe(x, y) \wedge \neg sobre(x, y) \rightarrow \exists z\, [encimaDe(x, z) \wedge encimaDe(z, y)]\, \}$

1. Eliminar implicadores y coimplicadores:
$\forall x\, \forall y\, \{\, \neg[encimaDe(x, y) \wedge \neg sobre(x, y)] \vee \exists z\, [encimaDe(x, z) \wedge encimaDe(z, y)]\, \}$

2. Normalizar negadores:
$\forall x\, \forall y\, \{\, [\neg encimaDe(x, y) \vee \neg\neg sobre(x, y)] \vee \exists z\, [encimaDe(x, z) \wedge encimaDe(z, y)]\, \}$
$\forall x\, \forall y\, \{\, [\neg encimaDe(x, y) \vee sobre(x, y)] \vee \exists z\, [encimaDe(x, z) \wedge encimaDe(z, y)]\, \}$

3. Normalizar variables:
4. Eliminar cuantificadores existenciales:

$\forall x \forall y \{ [\neg encimaDe(x, y) \vee sobre(x, y)] \vee [encimaDe(x, f(x, y)) \wedge encimaDe(f(x, y), y)] \}$

5. Forma prenexa:
6. Eliminar cuantificadores universales:

$\neg encimaDe(x, y) \vee sobre(x, y) \vee [encimaDe(x, f(x, y)) \wedge encimaDe(f(x, y), y)]$

7. FNC de la matriz de la fórmula:

$[\neg encimaDe(x,y) \vee sobre(x,y) \vee encimaDe(x,f(x,y))] \wedge [\neg encimaDe(x,y) \vee sobre(x,y) \vee encimaDe(f(x,y),y)]$

8. Extracción de cláusulas: eliminación de conjunciones

C_1: $\neg encimaDe(x, y) \vee sobre(x, y) \vee encimaDe(x, f(x, y))$

C_2: $\neg encimaDe(x, y) \vee sobre(x, y) \vee encimaDe(f(x, y), y)$

9. Normalización de variables:

C_1: $\neg encimaDe(x_1, y_1) \vee sobre(x_1, y_1) \vee encimaDe(x_1, f(x_1, y_1))$

C_2: $\neg encimaDe(x_2, y_2) \vee sobre(x_2, y_2) \vee encimaDe(f(x_2, y_2), y_2)$

6.- *Obtener, por el método del Cuadro, el valor de verdad de la siguiente fórmula proposicional:*

$$\mathbf{(p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r) \vee (\neg p \wedge q) \vee (\neg q \wedge \neg r)}$$

Simplificamos, quedando en FND:

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

1) No es contradicción

2) No existen conjunciones elementales consistentes en un sólo literal

3) Descomponemos la fórmula en:

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

y aplicamos de nuevo el método a cada una de las partes:

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

2) $p \leftarrow F$

$F \vee (V \wedge q) \vee (\neg q \wedge \neg r)$

$F \vee q \vee (\neg q \wedge \neg r)$

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

$q \leftarrow F$

$F \vee (V \wedge \neg r)$

$F \vee \neg r$

$\neg r \Rightarrow$ **INDETERMINACIÓN**

7.- *Determinar, por el método de Davis-Putnam, el valor de verdad de la fórmula proposicional:*

$$\mathbf{(p \vee q \vee r) \wedge (p \vee \neg q) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q)}$$

Ya está en FNC

1) No es tautología

2) No existen disyunciones elementales consistentes en un sólo literal

3) No existen variables que aparezcan sólo afirmadas o negadas

4) Elegimos la variable proposicional «*p*» y descomponemos:

B: $(p \vee q \vee r) \wedge (p \vee \neg q)$

C: $(\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q)$

D: $\varnothing$

quedando:

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

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

y aplicamos de nuevo el método:

1) No es tautología

2) $\neg q \overset{V}{\leftarrow}$ V $\quad (F \vee r \vee \neg r) \wedge (F \vee V \vee r) \wedge (F \vee V \vee \neg r) \wedge V$

$(r \vee \neg r) \wedge V \wedge V \wedge V$

$r \vee \neg r$

$V \quad \Rightarrow$ **INDETERMINACIÓN**

8.- *Determinar el valor de verdad de la fórmula proposicional utilizando el Método de Davis-Putnam:*

$$\mathbf{\neg p \wedge (p \vee q \vee r) \wedge (\neg q \vee r \vee \neg s) \wedge (\neg q \vee \neg r)}$$

Ya está en FNC

1) No es tautología

2) $\neg p \overset{V}{\leftarrow}$ V $\quad V \wedge (F \vee q \vee r) \wedge (\neg q \vee r \vee \neg s) \wedge (\neg q \vee \neg r)$

$(q \vee r) \wedge (\neg q \vee r \vee \neg s) \wedge (\neg q \vee \neg r)$

3) $\neg s \overset{V}{\leftarrow}$ V $\quad (q \vee r) \wedge (\neg q \vee r \vee V) \wedge (\neg q \vee \neg r)$

$(q \vee r) \wedge V \wedge (\neg q \vee \neg r)$

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

4) Elegimos *q* , descomponemos:

B: (q ∨ r)

C: (¬q ∨ ¬r)

D: ∅

quedando: r ∨ ¬r

∨ ⇒ **INDETERMINACIÓN**

5. TRABAJOS COMPLEMENTARIOS

Implementar alguno de los métodos para obtener el valor de verdad de una fórmula proposicional bien formada a partir de formas normales:

- Formas Normales Conjuntiva y Disyuntiva

- Método del Cuadro

- Método de Davis-Putnam

NOTA: Se puede utilizar cualquier lenguaje de programación. Se podrán obtener algunas ventajas si se utiliza algún lenguaje de programación declarativo, como por ejemplo LISP o PROLOG.

6. LA LÓGICA EN LA VIDA

En un pueblo se celebran todos los años las fiestas de «moros y cristianos». Tras varios días de fiesta y de batallas, siempre acaban siendo vencidos los moros. Cansados de perder año tras año, los componentes del bando *moro* decidieron mentir siempre; los *cristianos*, por contra, decidieron decir siempre la verdad.. En este contexto, un visitante, que unas veces mienten y otras dicen la verdad, aprovechó los días de fiestas para desvalijar las viviendas. Se realizaron varias detenciones. ¿ Podrías ayudar al juez ? :

1. Se detuvo a tres habitantes. Se sabía que uno era cristiano, otro moro y el tercero el ladrón; pero no se sabía quién era quién. Se hicieron las siguientes declaraciones: *A* dijo “Yo soy el ladrón”; *B* declaró “Yo no soy el ladrón”; y *C* manifestó “Yo no soy cristiano”. ¿ Sabrías deducir quién es el ladrón ?

2. En otra oportunidad volvieron ser detenidos tres habitantes: un cristiano, un moro y el ladrón. Pero no se sabía quién era cada uno Durante el interrogatorio *A* declaró que *C* era moro, y *B* declaró que *A* era cristiano. A *C* le preguntaron qué era y respondió: “Yo soy el ladrón”. ¿Quién era el ladrón, quién era el cristiano y quién era el moro?

3. Otro día se detuvo a un habitante, que era el ladrón, pero los oficiales no lo sabían. Se procedió al interrogatorio e hizo una declaración falsa. Inmediatamente le condenaron. ¿ Qué declaración hizo ?

4. Los oficiales, en otra ocasión, detuvieron a otro habitante, que aunque era el ladrón, los oficiales lo ignoraban. Esta vez el ladrón hizo una declaración auténtica, y también fue condenado. ¿ Cuál fue la declaración ?

5. En otro juicio teníamos de nuevo a tres acusados: un ladrón, un cristiano y un moro; pero no sabían quién era qué. El juez preguntó a A “¿Eres tú el ladrón?”. A contestó (sí o no). A continuación el juez preguntó a B “¿Dijo A la verdad?”. B respondió (otra vez sí o no). En ese momento A dijo “C no es el ladrón”. El juez respondió “Ya lo sabía. ¡Y ahora sé quién es el ladrón!”. ¿Quién era el ladrón?

Capítulo **8**

TÉCNICAS DE DEMOSTRACIÓN AUTOMÁTICA

Llegados a este punto estudiaremos procedimientos para construir demostraciones que sean adecuados para su realización utilizando el ordenador, y así de esta manera poder implementarlos. Para ello, en este tema vamos a ver una serie de técnicas para resolver problemas deductivos de manera automática, como paso previo a la Programación Lógica. La mayoría de procedimientos descritos en la literatura clásica son no-deterministas, como hemos podido comprobar; muchos de ellos son capaces de proporcionar cortas demostraciones cuando utilizamos conocimiento e inteligencia, es decir son usados por humanos, pero pueden llegar a ser extraordinariamente infinitos si se usan de manera puramente mecánica. Sin embargo, algunos pueden ser adaptados a usos mecánicos. Para ello, tras definir una serie de conceptos, como Universo y Base de Herbrand, Sustitución y Unificación, desarrollaremos la Regla de Resolución, una de las reglas más importantes de la lógica, desde el punto de vista de su aplicación a la informática.

En los capítulos anteriores, hemos desarrollado la base teórica de la lógica de primer orden, la cual nos sirve como apoyo para desarrollar una serie de procedimientos para la decisión de validez de fórmulas. Los procedimientos que vamos a describir aquí forman una parte importante en el entorno de la Inteligencia Artificial, en donde la utilización de la lógica se aplica a la resolución de problemas deductivos y a la contestación de preguntas. Dichos procedimientos se van a denominar *Métodos de Demostración Automática de Teoremas*, los cuales también se aplican en particular a las técnicas de verificación formal de programas y a la formalización de sistemas de programación lógica como PROLOG. Estos procedimientos que veremos pueden no terminar debido al grado de

indecibilidad del cálculo de predicados, pero esto no es relevante puesto que hay un gran número de fórmulas decidibles que justifican el desarrollo de éstos métodos.

En el Cálculo de Proposiciones, una fórmula con *n* variable proposicionales distintas, con valores de verdad en el conjunto {*V*, *F*}, tiene 2^n interpretaciones (un número finito); por ello decíamos que es decidible. Por otro lado, en el Cálculo de Predicados aparece el concepto de variable de término, la cual puede tomar valores en un determinado dominio o universo; como puedo tener infinitos dominios y no puedo evaluar la fórmula para todos estos infinitos dominios, decíamos que era indecicible. Aunque, posteriormente afirmabamos que era semidecidible, es decir, existen procedimientos en los que en el caso de que la fórmula sea válida lo resuelven en tiempo finito. Estos procedimientos se basan en introducir y trabajar sobre un dominio especial, el *Universo de Herbrand*.

Pasamos a continuación a desarrollar en este capítulo, la base teórica de los métodos de demostración automática, constituida y basada principalmente en el teorema de Herbrand y en el Principio de Resolución de Robinson, que constituyen junto con el algoritmo de unificación la base teórica que permiten la construcción de algoritmos. Se establece que un teorema está constituido por una serie de premisas y una conclusión que es deducible a partir de ellas. Gracias a la formalización en LPO de dichos teoremas y a la ayuda del teorema de deducción, el demostrar que una conclusión de una estructura deductiva es demostrable, se transforma en estudiar la validez de la fórmula equivalente a dicha estructura; luego en este sentido, un algoritmo de decisión permite comprobar si un teorema es demostrable y en particular veremos que lo es, comprobando si con la negación de la conclusión la fórmula es refutable.

Se establece que las fbf con las que vamos a trabajar de ahora en adelante, deben estar en *forma clausal.*

1. BASES TEÓRICAS

Partimos de una estructura deductiva P_1, P_2, ... $P_n \Rightarrow Q$ y tenemos que demostrar que $P_1 \wedge P_2 \wedge ... \wedge P_n \wedge \neg Q$ es insatisfactible, pues sabemos que si existiera un modelo de dicha fórmula sería un contraejemplo de la deducción y estaríamos ante una estructura deductiva no válida.

El comprobar que una fórmula es insatisfactible exige verlo para todas las interpretaciones posibles con lo cual, es tan extenso como comprobar la validez. Entonces, para reducir este estudio, Herbrand[1] planteó un modelo para el análisis de validez de una fórmula basado en la insatisfactibilidad; y de lo que se trata es de definir

[1] J. Herbrand. "Recherches sur la théorie de la démonstration". PhD Thesis, University of Paris, Traveaux de la Societé des Sciences de Varsovia, nº 33, 1930.

un universo determinado que constituya un dominio abstracto y estudiar si en él la fórmula es satisfactible o no y comprobar que si no es satisfactible en este universo entonces tampoco lo será en un dominio en particular. Pasamos a definir dicho universo que se llamará Universo de Herbrand (lo denotaremos para abreviar por UH).

Universo de Herbrand : el UH asociado a una fórmula es el conjunto de todos los términos sin variables que se pueden construir con las constantes y los símbolos de función del alfabeto de la fórmula.

El UH se crea a partir de las letras de constante y función que hay en la fórmula y se define así:

Sean C = { todas las letras de constante}
F = { todas las letras de función}
al universo lo denotamos por **UH** y será:

$$H = H_0 \cup H_1 \cup H_2 \cup ... \cup H_i \cup ... = \lim_{i \to \infty} H_i$$

donde cada H_i se define por:

H_0 - conjunto C de las constantes de la fórmula; si $C = \varnothing$ entonces se designa una constante cualquiera '*a*'.

...

H_i - se construye a partir de H_{i-1}, H_{i-2}, ... y H_1 aplicando las letras de función de F a sus elementos, de todas las formas posibles.

Como vemos el conjunto H es abstracto ya que según las asignaciones que se realicen obtendremos distintos conjuntos H. Por otro lado, siempre que haya funciones, es decir $F \neq \varnothing$, el universo de Herbrand es infinito. A partir de este momento, y como ya se ha comentado en el capítulo anterior, solamente se utilizarán fórmulas escritas en forma clausal.

Ejemplo:

Obtener el UH de las siguientes cláusulas:

C_1: P(a)
C_2: $\neg$ P(x) $\vee$ Q(f(x))

Solución: C = { letras constantes} = {a}
F = { letras de función } = {f}

Se construye la serie H_i de la siguiente forma:
H_0 = {a}

H_1 se construye a partir de H_0 y aplicando f a los elementos de H_0 , y así sucesivamente:

H_1 = { f(a)}
H_2 = { f(f(a))}
...

luego $H = H_0 \cup H_1 \cup ....$

H = { a, f(a), f(f(a)), ... }

Átomo Básico : fórmula resultante de asignar elementos del universo de Herbrand a las variables de los predicados de un conjunto de cláusulas.

Base de Herbrand (BH) : la BH asociada a una fórmula está constituida por todas las fórmulas resultantes de sustituir las variables de los predicados atómicos por todos los elementos de *UH*, es decir, es el conjunto formado por los productos cruzados entre los símbolos de predicado del alfabeto de la fórmula y los elementos del UH asociado a dicha fórmula. Sería el conjunto de los átomos básicos.

1.1. INTERPRETACIONES DE HERBRAND.

El dominio *H* definido antes es una estructura de dominio en el que se puede definir una interpretación de la fórmula que da origen al *UH*, esta interpretación se llama *IH*. Para definirla, vamos a ver primero unas definiciones:

Instancia Básica (IB) : *IB* de una cláusula es el resultado de asignar a las variables de la misma elementos del *UH*.

Ejemplo:
Sean las cláusulas:
C_1: $P(x_1)$
C_2: $\neg P(f(x_2))$

El *UH* es *H* = { a, f(a), f(f(a)),...}

Las *IB* para las cláusulas:
C_1 ⇨ **P(a)** , **P(f(a))** , ...
C_2 ⇨ **¬P(f(a))** , **¬P(f(f(a)))** , ...

Con estas bases teóricas podemos construir una **Interpretación de Herbrand** (**IH**) de la siguiente forma:

a) El domino es el universo de Herbrand (*UH*)asociado a la fórmula.

b) Hagamos las siguientes asignaciones:

1º.- Una constante se sustituye por ella misma (por definición aparecen en *UH*).

2º.- Si tenemos una función *f* de *n* variables entonces:

$$f: \quad UH^n \rightarrow UH$$
$$t_1, \ldots t_n \rightarrow f(t_1, \ldots t_n)$$

3º.- Para interpretar los predicados, se les asigna significado a los átomos de la Base de Herbrand. Si la $BH = \{ P_1, P_2, \ldots P_m \ldots\}$, una interpretación de los predicados que constituyen la fórmula quedará totalmente definida al asignar *V* o *F* a los P_i.

Entonces una Interpretación sería, por ejemplo, la siguiente

$$I = \{ P_1, P_2, \neg P_3 \ldots \}$$

(ponemos la fórmula atómica afirmada cuando sea el valor *V* y negada cuando sea *F*), que corresponden a la tabla de valores siguiente:

P_1	P_2	P_3	...
V	V	F	...

Ejemplo:

Sean las cláusulas:

C_1: $P(x, g(y)) \vee R(y)$

C_2: $R(g(y)) \vee P(x, y)$

Obtener el *UH*, la *BH*, y la interpretación correspondiente de Herbrand.

Solución:

C = { letras de constante (no tiene) } = {a}

F = { letras de función } = {g}

El universo de Herbrand es:

***H* = { a, g(a), g(g(a)), g(g(g(a))), ... }**

Para obtener la *BH* hacemos todas las asignaciones posibles de los elementos de *UH* a las variables de los predicados *P(w, z)* y *R(z)* donde *w* y *z* representan las plazas de los predicados.

***BH* = { P(a,a), R(a), P(a,g(a)), P(g(a),a), P(g(a),g(a)), R(g(a)), P(a,g(g(a))), P(g(g(a)),a), P(g(g(a)), g(g(a))), ... }**

Definimos ahora la *IH* de la siguiente forma:

1º.- A la letra '*a*' se le asigna ella misma.

2º.- *g* : $UH \rightarrow UH$
$h \rightarrow g(h)$

3º.- A los elementos de *BH* se le asignan los siguientes significados:

I ={P(a,a), R(a), ¬P(a,g(a)), ¬P(g(a),a), P(g(a),g(a)), ¬R(g(a)), ¬P(a,g(g(a))), ..., ¬R(g(g(a))), ... }

Veamos con esta definición cómo es la fórmula dada:

Una instancia básica de la C_2 es $R(g(g(a))) \vee P(a,g(a))$ que según la definición de I es:

¬R(g(g(a))) y ¬P(a,g(a)) pertenecen a *I*, luego el valor de verdad de C_2' es F ∨ F = F entonces como ambos átomos son falsos según la interpretación de la base *BH* la cláusula es falsa y por tanto también lo será la conjunción con la 1ª cláusula. Como la fórmula está en Forma Clausal se presupone la cuantificación Universal, luego la fórmula no se satisface para esta interpretación.

Relación entre dominios y Universo de Herbrand

Dada una fórmula y una interpretación *I* de la misma en un dominio concreto *D,* podemos definir una interpretación *I** en el universo de Herbrand correspondiente a ella. Para ello:

1º.- Se da un dominio concreto y una interpretación de la fórmula dada.

2º.- Se construye el universo y la base de Herbrand para dicha fórmula.

3º.- Se va construyendo la interpretación de esa fórmula basada en dicha base de Herbrand y en la interpretación de partida.

Ejemplo 1:

Sea la fórmula, en forma clausal:

C_1: $P(x_1,g(y_1)) \vee R(y_1)$.

C_2: $R(g(x_2)) \vee P(x_2,y_2)$

Dicha fórmula se interpreta en el dominio D = {1, 2, 3} de la siguiente forma:

valores para las variables, la función y los predicados

x	y	g(x)	R(x)	P(x,y)
1	1	2	V	V
	2			V
	3			F
2	1	3	F	F
	2			F
	3			V
3	1	1	V	V
	2			F
	3			V

Ahora se construye el *UH* (ejercicio anterior):

H = { a, g(a), g(g(a)), g(g(g(a))), ... }

Nota : Por comodidad llamamos $g^1(a) = g(a)$, $g^2(a) = g(g(a))$, etc.

Construimos la *BH*:

BH = { P(a,a), R(a), P(a,g^1(a)), P(g^1(a), a), P(g^1(a), g^1(a)), R(g^1(a)), P(a,g^2(a)),P(g^1(a), g^2(a)), ... }

Con la interpretación dada construimos otra basada en la *BH* y lo hacemos de la siguiente manera:

- A la letra '*a*' le asignamos el valor 1 (luego se le asignará el 2 y el 3).

- P(a,a) $\Leftrightarrow$ P(1,1) $\leftarrow$ *V*
  R(a) $\Leftrightarrow$ R(1) $\leftarrow$ *V*
  P(a,g^1(a)) $\Leftrightarrow$ P(1,g^1(1)) $\Leftrightarrow$ P(1,2) $\leftarrow$ *V*
  ...

Por tanto una interpretación de la fórmula en el *UH* será:

I * = { P(a,a), R(a), P(a,g^1(a)), ¬P(g^1(a),a), P(g^1(a),g^1(a)), ¬R(g^1(a)), ... }

Para el valor de *a* = 2 y 3 se tendrán otras interpretaciones de *H* correspondientes a *I*.

Ahora evaluamos la fórmula según *I** para los valores de las variables *x*=*a*, *y*=*a* y resulta:

C_1: P(a,g(a)) $\vee$ R(a)
C_2: R(g(a)) $\vee$ P(a,a)

Ahora miramos cómo son estas fórmulas de acuerdo a los valores de la *BH*

P(a,g(a)) $\leftarrow$ V
R(a) $\leftarrow$ V
luego P(a,g(a)) $\vee$ R(a) $\leftarrow$ V

y lo mismo para C_2

R(g(a)) $\leftarrow$ F
P(a,a) $\leftarrow$ V
luego R(g(a)) $\vee$ P(a,a) $\leftarrow$ V

luego la fórmula es verdadera.

Ejemplo 2:

Sea la fórmula: C_1: P(a) $\vee$ Q(x_1)
C_2: R(x_2,y)

Se define una interpretación en el domino *D*= {1,2} de la siguiente forma:

x	y	P(x)	Q(x)	R(x,y)
1	1	V	F	V
	2			V
2	1	F	V	V
	2			V

Esta interpretación satisface a la fórmula pues *P(a)* $\vee$ *Q(x)* y *R(x,y)* son válidas en *D*.

Ahora calculamos el *UH* y la *BH*:

H = {a}
BH = { P(a), Q(a), R(a,a) }

Definimos una interpretación para este universo

I* = { P(a), ¬Q(a), R(a,a) }

y de acuerdo a esta interpretación la fórmula se satisface.

1.2. TEOREMA DE HERBRAND

El teorema de Herbrand reduce el estudio de la satisfacibilidad en la lógica de predicados al de la lógica proposicional. Como hemos visto al inicio de este capítulo, al aplicar refutación transformamos el estudio de la corrección de una estructura deductiva en el estudio de la insatisfacibilidad de la fórmula asociada a ella. Para ello, intentaremos demostrar que la valoración semántica de dicha fórmula es falsa, lo cual se cumplirá si y sólo si alguna de las subfórmulas (cláusulas) que forman la conjunción es falsa (recordemos que estabamos trabajando con fórmulas en forma clausal). Como las variables que aparecen en dicha fórmula están cuantificadas universalmente, para que sea falsa bastará que para algún elemento del Universo de Herbrand el valor de la subfórmula sea falso. Por tanto, será suficiente con encontrar una instancia básica de la cláusula que sea falsa. Por lo que nuestro problema queda reducido al estudio de las instancias básicas de las cláusulas.

La condición necesaria y suficiente para que una fórmula sea *insatisfacible* es que exista un conjunto finito de instancias de base de las cláusulas de la fórmula que sea insatisfacible

Resultados:

Si una fórmula es satisfactible entonces existe al menos una interpretación de Herbrand que la satisface

Una fórmula es insatisfactible si y sólo si es falsa para todas las interpretaciones de Herbrand

Los métodos que vamos a ver basados en este teorema detectan si una fórmula es insatisfacible, para ello buscan conjuntos no vacíos de instancias de base y se comprueban si son insatisfacibles, y por lo que indica el teorema basta encontrar uno de ellos para decidir que la fórmula lo sea o no.

2. MÉTODOS DE DEDUCCIÓN AUTOMÁTICA BASADOS EN EL TEOREMA DE HERBRAND

Estos métodos propuestos se basan en buscar conjuntos no vacíos de instancias de base y comprobar si son insatisfacibles. En general, cada método llevará la siguiente plantilla de trabajo:

1º.- Poner la fórmula en *forma clausal*.
2º.- Generar el *UH*.
3º.- Generar conjuntos de *IB*.
4º.- Generar combinaciones de las distintas instancias obtenidas para cada cláusula.
5º.- Comprobar satisfactibilidad mediante métodos.

Los métodos que vamos a estudiar son:
- Gilmore.
- Davis y Putnam.

2.1. MÉTODO DE GILMORE

Proceso:

1º.- Transformar la fórmula en disyunción de conjunciones de los átomos del Universo de Herbrand.

2º.- Si la fórmula es insatisfacible entonces aparecerá una contradicción en cada una de las conjunciones.

Ejemplo:

Estudiar la siguiente fórmula en forma clausal utilizando el método de Gilmore:
C_1: $P(x_1)$
C_2: $\neg P(f(x_2))$

Solución: Primero calculamos el UH

$$H = \{ a, f^1(a), f^2(a), f^3(a), \ldots \}$$

Calculamos las instancias base para cada cláusula:

$P(x_1)$	$\neg P(f(x_2))$
$P(a)$	$\neg P(f^1(a))$
$P(f^1(a))$	$\neg P(f^2(a))$
$P(f^2(a))$	$\neg P(f^3(a))$
...	...

Hacemos el paso 1 del proceso que es transformar la fórmula en disyunciones ya que de acuerdo con el teorema de Herbrand las conjunciones insatisfacibles pueden ser:

$P(a) \wedge \neg P(f^1(a))$
$P(a) \wedge \neg P(f^2(a))$
$P(f^1(a)) \wedge \neg P(f^1(a))$
$P(f^1(a)) \wedge \neg P(f^2(a))$
etc.

Vemos que la que es contradicción es $P(f^1(a)) \wedge \neg P(f^1(a))$

En este caso no hay que formar una conjunción de disyunciones pues hay una única conjunción, y vemos que algunas de las interpretaciones H son contradicciones, luego la fórmula es insatisfacible.

2.2. MÉTODO DE DAVIS - PUTNAM (Generalizado)

El método de Gilmore es aplicable cuando el número de cláusulas no es demasiado grande, pues el proceso de formación de conjunciones y el de eliminación de cada una de ellas para encontrar la contradicción puede ser muy largo.

Davis y Putnam proponen una método simplificado que permite decidir si un conjunto de instancias base es insatisfacible. Para ello definen unas reglas que permiten pasar de la conjunción de cláusulas inicial a otras más sencillas y se establecerá que si éstas son insatisfacible también lo será el conjunto inicial. Este método ya visto en el tema anterior para fórmulas proposicionales, es ahora generalizado para trabajar con fórmulas del cálculo de predicados.

Proceso :

a) Se recorre la fórmula identificando las tautologías y suprimiendo las que aparezcan. Para ello se elimina del conjunto de instancias aquellas en las que aparezca un literal y su negado.

b) Se identifican los literales aislados y se aplica uno de los casos siguientes:

b1.- si aparecen una instancia con el literal L y otra con el literal $\neg L$ el proceso termina y la fórmula es insatisfacible.

b2.- Si aparece una instancia constituida por el literal aislado L, se elimina esta instancia y todas aquellas en las que aparezca L; aquellas instancias en las que aparezca el literal $\neg L$, desaparece dicho literal de ellas. Si esta fórmula es insatisfacible también lo es la primera.

b3.- Se prescinde de todas las instancias en que aparece el literal puro. Se dice que un literal es puro cuando aparece sólo de una forma o bien como L o como $\neg L$.

En cada una de estos dos últimos casos, si desaparecen todas las cláusulas entonces la fórmula es satisfacible.

c) Aplicar la fórmula de descomposición:

Elegir uno de los literales *L*.
Dividir la fórmula de instancias respecto del literal *L* elegido de la siguiente forma:

A = conjunto de instancias que contienen *L*
B = conjunto de instancias que contienen ¬*L*
C = conjunto de instancias que no contienen *L* de ninguna forma
y se reemplaza:

$$F = [\ \wedge (A_i \vee B_j)\ ] \wedge C$$

d) Repetir el proceso hasta encontrar una situación de terminación que se presenta o bien cuando se llega a la fórmula vacía sin haber encontrado contradicción y entonces la fórmula es satisfacible o por obtener una contradicción por lo que la fórmula sería insatisfacible.

Ejemplo:

Comprobar que es insatisfacible el conjunto de cláusulas siguiente, por el método de Davis-Putnam:

C_1: ¬ P(x) ∨ Q(f(x),x)
C_2: P(a)
C_3: ¬Q(y,z)

Solución:

Partimos del conjunto de instancias.

1.- ¬P(a) ∨ Q(f(a),a), P(a), ¬Q(a,a)

Se aplica la regla del literal aislado para *P(a)*, y obtenemos el conjunto *Q(f(a)),a), ¬Q(a,a)*. Como en esta fórmula cualquiera de los dos literales son aislados entonces se obtiene aplicando dos veces la regla *b2)* el conjunto vacío, luego la fórmula es satisfacible.

2.- ¬P(a) ∨ Q(f(a),a), P(a), ¬Q(a, f(a))

Se aplica la regla *b2)* sobre *P(a)* y resulta *Q(f(a),a),¬Q(a,f(a))*. Otra vez se aplica *b2)* dos veces sobre esta fórmula y resulta la fórmula vacía; luego la fórmula es satisfacible.

3.- ¬P(a) ∨ Q(f(a),a), P(a), ¬Q(f(a),a)

Se aplica la regla *b2*) y se obtiene Q(f(a),a), ¬Q(f(a),a). A esta fórmula se aplica *b1*) y se obtiene una contradicción L y ¬L luego la fórmula es insatisfacible entonces la fórmula original es insatisfacible.

3. MÉTODO DE UNIFICACIÓN Y RESOLUCIÓN

La resolución es una de las reglas de inferencia más importantes y se basa en el siguiente resultado de la lógica:

Si de un conjunto de premisas se deduce una contradicción, no puede existir un modelo de dicho conjunto de premisas.

En efecto, si $P_1, P_2, ..., P_n \Rightarrow B \wedge \neg B$ es un deducción correcta entonces todo modelo de $P_1, ..., P_n$ debe satisfacer $B \wedge \neg B$. Pero como no existe ninguna interpretación que satisfaga $B \wedge \neg B$, tampoco podrá existir modelo de $P_1, ..., P_n$.

La mejor manera de entender la Regla de Resolución es dando primero la formulación para cláusulas básicas sin variables; posteriormente generalizaremos para el caso en que las cláusulas contengan variables. Para ello, previamente deberemos dar un par de conceptos : unificación y sustitución.

3.1. SUSTITUCIONES Y UNIFICACIÓN

Llamamos *Unificación* al proceso de encontrar sustituciones de términos que hagan idénticas ciertas expresiones.

Ejemplo:

Si tenemos las expresiones $\forall x\ (P(x) \rightarrow Q(x))$ y $P(a)$, para aplicar la regla del MP tenemos que realizar la sustitución del término x por a, con esto tendríamos que son idénticas las expresiones $P(x)$ y $P(a)$, y la aplicación de la regla será inmediata.

Con esta idea clara, debemos explicar detenidamente el concepto de *sustitución*.

Sustitución : proceso por el cual se sustituyen términos ($t_1, t_2, ..., t_n$) en lugar de variables ($v_1, v_2, ..., v_n$). La sustitución la podemos representar por un conjunto finito de pares ordenados:

$$s = \{ t_1/v_1, t_2/v_2, ..., t_n/v_n \}$$

Recordemos que los términos que pueden aparecer en nuestras expresiones son símbolos de variables, de constantes o de función.

IMPORTANTE:

- El concepto "ordenado" significa que las sustituciones se realizan en el orden en que aparecen en el conjunto.

- Se insiste en que una sustitución debe ser tal que todas las ocurrencias de una variable sean sustituidas por el mismo término.

- Además, ninguna variable puede ser reemplazada por una función que contenga esa misma variable.

- Las sustituciones siempre se hacen de poner términos donde había variables.

Ejemplo:

Sea la expresión *P(x, f(y), a)*. Algunas de las sustituciones que se pueden realizar y las expresiones que se obtendrían son:

s_1 = { z/x, w/y }	P(z, f(w), a)
s_2 = { a/y }	P(x, f(a), a)
s_3 = { g(z)/x, a/y }	P(g(z), f(a), a)
s_4 = { c/x, a/y }	P(c, f(a), a)

Como se ve siempre se sustituye una variable por un término en general (constante, variable o función). NUNCA CONSTANTE O FUNCIÓN POR TÉRMINOS.

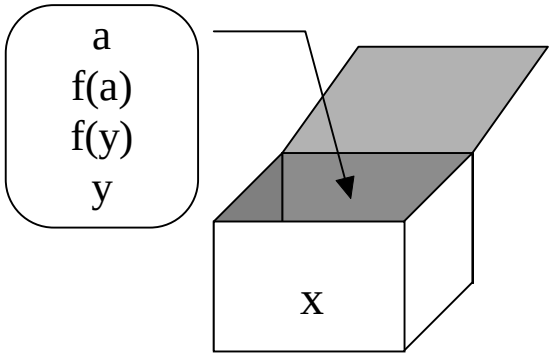


Ilustración 1: Sustitución → variable por término

Composición de sustituciones:

La composición de sustituciones s_1 y s_2 se representa por $\boldsymbol{s_1 \circ s_2}$ que es la sustitución obtenida aplicando s_2 a los términos (del numerador) de s_1 y añadiéndole

después los pares de s_2 que tengan variables (en el denominador) que no aparezcan en s_1 como variables a sustituir.

Ejemplo:

s_1 = { g(x, y)/z }
s_2 = { a/x, b/y, c/w, d/z }

s_1 o s_2 = { g(a, b)/z, a/x, b/y, c/w }

Propiedades. Si *E* es una expresión cualquiera y s_1, s_2, s_3 son sustituciones, entonces:

- (E s_1) s_2 = E (s_1 o s_2)
- La composición de sustituciones es asociativa: (s_1 o s_2) o s_3 = s_1 o (s_2 o s_3)
- Las sustituciones no son, en general, conmutativas: s_1 o $s_2 \neq s_2$ o s_1

Unificación : decimos que un conjunto $\{E_i\}$ de expresiones (i=1,..., n) es **unificable** si existe una sustitución s tal que $E_1 s = E_2 s = ... = E_n s$.

En este caso se dice que s es el **unificador** de $\{E_i\}$, ya que su aplicación convierte el conjunto $\{E_i\}$ en otro con un solo elemento.

Ejemplos:

1.- Sean las expresiones $E_1 = P(x, f(y), b)$ y $E_2 = P(x, f(b), b)$. Sea $s_1 = \{a/x, b/y\}$ una sustitución que aplicada a E_1 y E_2 nos da:
$E_1 s_1$ = P(x, f(y), b) {a/x, b/y} = P(a, f(b), b)
$E_2 s_1$ = P(x, f(b), b) {a/x, b/y} = P(a, f(b), b)

Luego s_1 es un unificador para E_1 y E_2.

Consideremos ahora la sustitución $s_2 = \{ b/y\}$ aplicada a E_1 y E_2
$E_1 s_2$ = P(x, f(y), b) {b/y} = P(x, f(b), b)
$E_2 s_2$ = P(x, f(b), b) {b/y} = P(x, f(b), b)

Luego s_2 también es un unificador para E_1 y E_2.

2.- Las expresiones $E_1 = P(f(x), a)$ y $E_2 = P(y, f(z))$ no son unificables, ya que el segundo argumento de las expresiones no se puede unificar, es decir, no hay ninguna sustitución que las haga iguales ya que *a* no puede unificar con *f(z)*.

El unificador por lo general no es único. En el primer ejemplo anterior hemos visto que podemos definir, al menos, dos unificadores. Debemos ahora pensar cuál es el unificador que más nos conviene. Si analizamos las dos expresiones unificadas obtenidas, *P(a,f(b),b)* y *P(x f(b),b)*, la primera reduce una posterior unificación ya que el

primer argumento está instanciado a una constante *a* mientras que en la segunda expresión tenemos una variable *x*. Por lo tanto nos interesa un unificador lo más simple posible para hacer que se puedan unificar un conjunto lo más extenso posible de expresiones unificables. A este le llamaremos el **Unificador Más General** (UMG).

Algoritmo para encontrar el unificador más general de dos expresiones

Entrada:

- Dos expresiones escritas en forma de listas de la forma:
  P(x, f(y), c) = [P, x, [f, y], c]

Salida:

- Unificador mas general de un conjunto de dos expresiones unificables, o
- FALLO, si las expresiones no son unificables.

Procedimiento recursivo **UNIFICAR: UMG (E1, E2)**

1. Si alguno de E_1 o E_2 es un átomo (es decir, símbolo de predicado, de función, de constante, de negación o de variable) intercambiar los argumentos E_1 y E_2, si es necesario, de modo que E_1 sea un átomo y entonces:
   - 1.1. Si E_1 y E_2 son idénticos → NADA.
   - 1.2. Si E_1 es un símbolo de variable hacer:
     - si aparece E_1 en E_2 → FALLO
     - sino → { E_2/E_1 }
   - 1.3. Si E_2 es un símbolo de variable → { E_1/E_2 }
   - 1.4. → FALLO

   fin 1

2. Si no, hacer:
   - 2.1. Z_1 : = UMG (primero (E_1), primero (E_2))
   - 2.2. Si Z_1 = FALLO → FALLO
   - 2.3. Z_2 : = UMG (resto(E_1) Z_1, resto(E_2) Z_1)
   - 2.4. Si Z_2 = FALLO → FALLO
   - 2.5. → (Z_1 o Z_2)

   fin 2

Fin

Normalmente usaremos UMG para descubrir si *un literal puede concordar con otro.* El proceso de encontrar una expresión con otra que se toma como patrón se llama confrontación con patrones (“pattern matching”) y tiene un papel importante en IA. Este proceso que hemos visto es más general que la confrontación de patrones.

Ejemplo:

Encontrar el Umg de las expresiones $E_1 = P(x, y)$ y $E_2 = P(x, z)$

Solución: Ponemos las expresiones como listas: UMG ([P, x, y], [P, x, z])

1ª llamada:
Paso 1: no es átomo *[P, x, y]* ni *[P, x, z]* entonces
Paso 2: Z_1 = UMG(primero(E_1), primero(E_2)) = UMG(P, P)

2ª llamada: UMG(P, P)
Paso 1: sí es átomo *P* y *P* y son iguales luego devolver NADA

$Z_1 \leftarrow$ NADA

Z_2 = UMG(resto(E_1)Z_1, resto(E_2)Z_1) = UMG([x, y]NADA, [x, z]NADA) =UMG([x, y], [x, z])

3ª llamada: UMG([x, y], [x, z])
Paso 1: ni *[x, y]* ni *[x, z]* son átomos entonces
Paso 2: Z_1 = UMG(primero([x, y]), primero([x, z])) = UMG(x, x)

4ª llamada: UMG(x, x)
Paso 1: *x* y *x* son átomos e iguales, luego $\rightarrow$ NADA

$Z_1 \leftarrow$ NADA

Z_2=UMG(resto(E_1)Z_1,resto(E_2)Z_1)=UMG(yNADA, zNADA)=UMG(y,z)

5ª llamada: UMG (y, z)
Paso 1: *y* y *z* son átomos distintos, *y* es una variables y como no aparece *y* en *z* $\rightarrow$ { E_2/E_1 } = { z/y }

$Z_2 \leftarrow$ { z/y }

Devolver NADA o { z/y } = { z/y }

$Z_2 \leftarrow$ { z/y }

Devolver NADA o { z/y } = { z/y }

UMG([P, x, y], [P, x, z]) = { z/y }

Unificación múltiple:

El problema de la unificación múltiple lo podemos ver como una secuencia de unificaciones binarias, así podemos definir una secuencia de sustituciones, cada una obtenida por el algoritmo anterior UMG de manera que si tenemos un conjunto de expresiones $\{E_1, ..., E_n\}$ y

s_1 es el UMG de E_1 y E_2
s_2 es el UMG de $E_1 s_1$ y $E_3 s_1$
...

s_{n-1} es el UMG de $E_1\ s_1\ s_2\ ...\ s_{n-2}$ y $E_n\ s_1\ s_2\ ...\ s_{n-2}$

Podemos decir que s_1 o s_2 o ... o s_{n-1} es el UMG del conjunto $\{E_1, ..., E_n\}$.

Ejemplo:

Encontrar el Unificador Más General de las expresiones $E_1 = P(x, a)$, $E_2 = P(f(z), y)$ y $E_3 = P(f(b), w)$ mediante unificación múltiple con el algoritmo UMG.

Solución:

Ponemos las expresiones como listas { [P, x, a], [P,[f, z], y], [P, [f, b], w] } y aplicamos el algoritmo UMG:

s_1 = UMG([P, x, a], [P,[f, z], y]) = {f(z)/x, a/y}

s_2 = UMG([P,x , a] {f(z)/x, a/y}, [P, [f, b], w] {f(z)/x, a/y}) =
= UMG([P,[f, z], a], [P, [f, b], w]) = {b/z, a/w}

Por tanto el UMG será:

s = s_1 o s_2 = {f(z)/x, a/y} o {b/z, a/w} = **{f(b)/x, a/y, b/z, a/w}**

3.2. REGLA DE RESOLUCIÓN

La Regla de Resolución[2] dice:

> «Dadas dos instancias o cláusulas básicas que contienen una el literal L y otra $\neg L$, es decir, de la forma $L \vee A_1$, y $\neg L \vee A_2$, puede deducirse de ambas la fórmula $A_1 \vee A_2$. Esta fórmula se llama cláusula **resolvente**, y a las fórmulas a las que se les aplica, cláusulas **padres**.»

La cláusula resolvente se calcula tomando la disyunción de las dos cláusulas y eliminando de ellas el par de complementarios L y $\neg L$.

Veamos informalmente las diferentes posibilidades para determinar si la resolución es creíble. Primero: supongamos que L es verdadera, y por tanto $\neg L$ es falsa; si $\neg L$ es falsa de la 2ª expresión se deduce que A_2 debe ser verdadera, y por ello $A_1 \vee A_2$ es verdadera. Segundo: supongamos que L es falsa, entonces de la 1ª expresión deducimos que A_1 es verdadera y por tanto $A_1 \vee A_2$ también será verdadera. Luego se

[2] Robinson, J.A. "A Machine-oriented Logic Based on the Resolution Principle". Journal of the ACM (Association for Computing Machinery), vol. 12, nº 1, 1965, pp. 23-41

puede concluir que el resolvente $A_1 \vee A_2$ debe ser verdadero puesto que $L \vee A_1$ y $\neg L \vee A_2$ son verdaderos.

Cláusulas Padre

$[L \vee A_1 \vee A_2 \vee ... \vee A_n]$ y $[\neg L \vee B_1 \vee B_2 \vee ... \vee B_m]$

$\Downarrow$

Cláusula Resolvente

$[A_1 \vee ... \vee A_n \vee B_1 \vee ... \vee B_m]$

Regla de Resolución sin variables

Se puede demostrar fácilmente, por deducción natural, que dicha estructura deductiva es correcta:

$$L \vee A_1, \neg L \vee A_2 \implies A_1 \vee A_2$$

- 1	$L \vee A_1$	
- 2	$\neg L \vee A_2$	
3	$A_1 \vee L$	CD 1
4	$\neg A_1 \rightarrow L$	DfD_1 3
5	$L \rightarrow A2$	DI 2
6	$\neg A1 \rightarrow A2$	Sil 4, 5
7	$A1 \vee A2$	DfD_1 6

Ejemplos triviales:

Cláusulas padre	Resolvente	
P y $\neg P \vee Q$	Q	Silogismo Disyuntivo
$P \vee Q$ y $\neg P \vee Q$	Q	
$P \vee Q$ y $\neg P \vee \neg Q$	$Q \vee \neg Q$ $P \vee \neg P$	Aquí hay dos resolventes posibles. Ambos casos son tautologías.
$\neg P$ y P	NADA	La cláusula vacía es signo de contradicción
$P \vee Q$ y $\neg P \vee R$	$Q \vee R$	

Si se aplica la regla a una contradicción, es decir, cuando se llega a deducir *B* y ¬*B*, la cláusula resolvente de ambas es la cláusula vacía. Esta propiedad puede enunciarse así:

Si aplicando la regla de resolución a un conjunto de cláusulas se obtiene la cláusula vacía, el conjunto de cláusulas es ***insatisfactible****.*

La idea es aplicar la regla de resolución al conjunto de cláusulas hasta que, o bien se llega a la cláusula vacía y entonces el conjunto inicial es insatisfactible, o bien no aparece la cláusula vacía y es satisfactible.

Pero veamos ahora como se generaliza esta regla para el caso de que tengamos cláusulas con variables. Necesitamos encontrar una sustitución que aplicada a las cláusulas padres haga que estas contengan literales complementarios.

Ejemplo:

$[P(x, y) \vee Q(x)]$ y $[\neg Q(a) \vee R(z)]$

Estas dos cláusulas no contienen átomos complementarios, ya que Q(x) y ¬Q(a) no lo son; pero si que podemos encontrar una sustitución {a/x} que aplicada a ambas cláusulas hace que obtengamos dos átomos que sí son complementarios. Así:

$[P(x, y) \vee Q(x)] \{a/x\} = [P(a, y) \vee Q(a)]$

$[\neg Q(a) \vee R(z)] \{a/x\} = [\neg Q(a) \vee R(z)]$

que se resuelven en

$[P(a,y) \vee R(z)]$

«Si tenemos las cláusulas padre $\{L_i\}$ y $\{M_i\}$. Sean $\{l_i\}$ subconjunto de $\{L_i\}$ y $\{m_i\}$ subconjunto de $\{M_i\}$ tales que existe un UMG *s* para $\{l_i\}$ y $\{\neg m_i\}$. Obtendremos la resolvente $\{ \{L_i\}\text{-}\{l_i\} \}\ S \cup \{ \{M_i\}\text{-}\{m_i\} \}\ s$ »

Cláusulas Padre

$[l_i \vee A_1 \vee A_2 \vee ... \vee A_n]$ y $[m_i \vee B_1 \vee B_2 \vee ... \vee B_m]$

$\Downarrow \quad s = \text{UMG}(l_i, \neg m_i)$

Cláusula Resolvente

$[A_1 \vee ... \vee A_n \vee B_1 \vee ... \vee B_m]\ s$

Regla de Resolución con variables

Ejemplos triviales:

Cláusulas padre	Resolvente	Sustitución
$P(x) \vee Q(a)$ y $\neg Q(a) \vee R(z)$	$P(x) \vee R(z)$	
$Q(x)$ y $\neg Q(a)$	NADA	$\{a/x\}$
$P(x,y) \vee Q(x,b)$ y $\neg Q(a,y) \vee R(y)$	$P(a,b) \vee R(b)$	$\{a/x, b/y\}$

El resolvente no es único, ya que puede haber más de una manera de escoger $\{l_i\}$ y $\{m_i\}$. Esta elección es importante, y da lugar a las *estrategias de resolución* que veremos más adelante. Hay por tanto dos decisiones a tomar al hacer la resolución:

- la selección de las cláusulas a unificar y

- la selección de los átomos a unificar dentro de esas cláusulas.

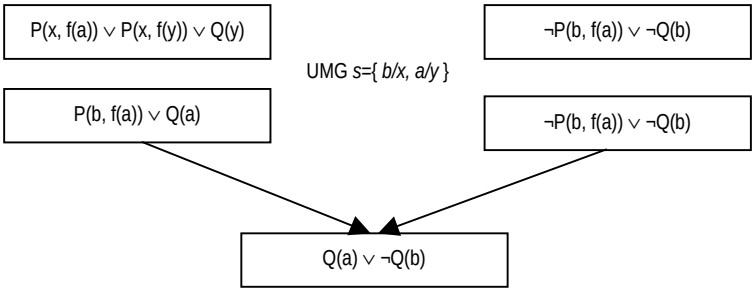
Ejemplo:

$$\{ P(x, f(a)) \vee P(x, f(y)) \vee Q(y) \} \text{ y } \{ \neg P(b, f(a)) \vee \neg Q(b) \}$$

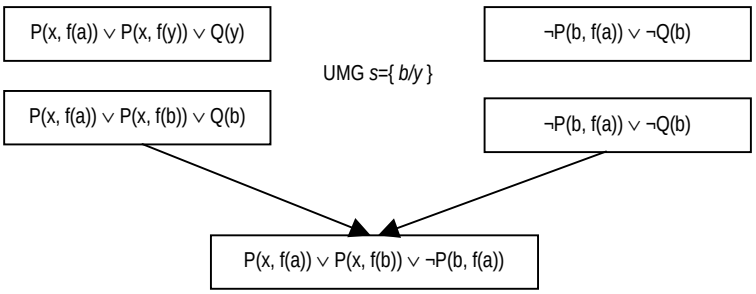
Con *{ a/y }* unifico *P(x, f(a))* y *P(x, f(y))* en *P(x, f(a))*

Y con *{ b/x }* unifico *P(x, f(a))* y *¬P(b, f(a))*

Por resolución obtengo la resolvente $Q(a) \vee \neg Q(b)$ con el UMG S={ *b/x, a/y* }. Gráficamente quedaría:



Si el átomo a unificar hubiese sido el predicado Q tendríamos:



3.3. SISTEMAS DE REFUTACIÓN POR RESOLUCIÓN

Los sistemas basados en la resolución están concebidos para producir demostraciones por reducción al absurdo o **refutaciones**:

- Partimos de un conjunto de fórmulas bien formadas *S* a partir del cual deseamos demostrar cierta fbf *objetivo W*.

- Negamos la fbf objetivo ($\neg W$) y añadimos ésta al conjunto de fórmulas *S*

- Usando la resolución sobre este conjunto ampliado, intentaremos llegar a una contradicción, representada por la cláusula vacía *NADA*.

- Si hemos encontrado la cláusula vacía entonces podemos concluir que la fórmula objetivo *W* se deduce del conjunto premisas *S*.

Idea: convertir $S \wedge \neg W$ en cláusulas y demostrar que de ellas se sigue *NADA* (cláusula vacía), que significa contradicción. Es decir:

$$S \wedge \neg W = \text{Falso} \implies \neg(S \wedge \neg W) = \text{Verdadero} \implies S \rightarrow W = \text{Verdadero}$$

El argumento para justificar el proceso de demostración por refutación es:

«Si *W* se sigue de *S* entonces cualquier interpretación que satisfaga *S* satisface también *W* y no a $\neg W$, por lo tanto, ninguna interpretación puede satisfacer *S* unión $\neg W$. Un conjunto de fbf que no pueda ser satisfecho por interpretación alguna se llama **incompatible,** así, si W se sigue de S entonces S unión ¬W es incompatible. Se ve entonces que si la resolución se aplica repetidamente a un conjunto de cláusulas incompatibles se produce *NADA*.»

Algoritmo de refutación por resolución:

1.- Negar la fórmula que se va a probar ($\neg W$) y añadirlo a la lista de premisas (*S*). Estas fórmulas estarán escritas en forma clausal.

Cláusulas := $S \cup \neg W$

2.- Mientras haya una pareja resoluble de cláusulas hacer:
- escoger dos cláusulas distintas c_i y c_j
- calcular la resolvente r_{ij} de c_i y c_j
- Cláusulas := añadir r_{ij}
- si r_{ij}= *NADA* entonces el teorema es verdadero

3.- El teorema es falso

Normalmente resulta ilustrativo utilizar un diagrama en forma de árbol para registrar la manera en que las cláusulas se resuelven en el proceso de producción de la cláusula vacía. Esta estructura la vamos a llamar *grafo de derivación*. Los nodos de ese grafo están etiquetados con cláusulas. Cuando dos cláusulas c_i y c_j producen un resolvente r_{ij} se crea un nuevo nodo etiquetado con r_{ij} con arcos que los unen a c_i y c_j. Diremos que c_i y c_j son los padres de r_{ij} y, éste un descendiente de ellos. Una refutación por resolución se plantea como un *árbol de refutación* que tiene un nodo raíz etiquetado con *NADA*.

Ejemplo:

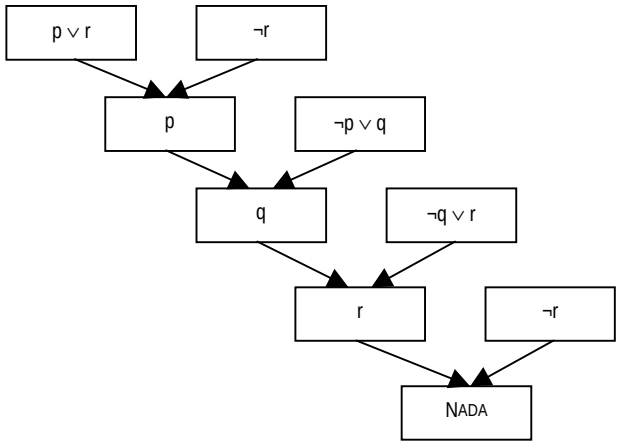
1. Utilizando el método de Refutación por Resolución demostrar el siguiente argumento del Cálculo de Proposiciones:

$$(p \rightarrow q) \rightarrow r, \neg p \vee q \Rightarrow r$$

Obtenemos la Forma Clausal de la fórmula correspondiente a la afirmación de las premisas y la negación de la conclusión:

$$[(p \rightarrow q) \rightarrow r] \wedge (\neg p \vee q) \wedge \neg r$$
$$[\neg(\neg p \vee q) \vee r] \wedge (\neg p \vee q) \wedge \neg r$$
$$[(\neg\neg p \wedge \neg q) \vee r] \wedge (\neg p \vee q) \wedge \neg r$$
$$[(p \wedge \neg q) \vee r] \wedge (\neg p \vee q) \wedge \neg r$$
$$(p \vee r) \wedge (\neg q \vee r) \wedge (\neg p \vee q) \wedge \neg r$$

C_1: $p \vee r$
C_2: $\neg q \vee r$
C_3: $\neg p \vee q$
C_4: $\neg r$



Al obtener la cláusula NADA, podemos concluir que el conjunto inicial de cláusulas es insatisfacible, y por tanto, que la conclusión se deduce de las premisas.

2. Utilizando el método de Refutación por Resolución demostrar el siguiente argumento:

Cualquiera que puede cantar es cantante
Los pájaros no son cantantes
Algún pájaro tiene buena voz
Luego, Alguien que tiene buena voz no puede cantar

Formalizamos:

$\forall x (R(x) \rightarrow C(x))$	R: poder cantar
$\forall x (P(x) \rightarrow \neg C(x))$	C: ser cantante
$\exists x (P(x) \wedge V(x))$	P: ser pájaro
$\Rightarrow \exists x (V(x) \wedge \neg R(x))$	V: tener buena voz

Convertimos a Forma Clausal:

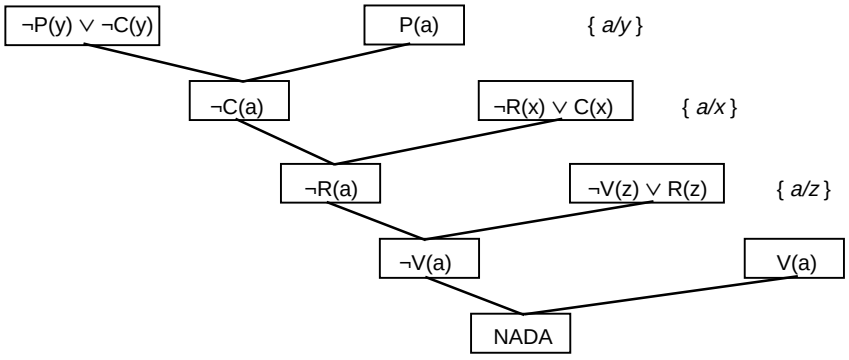
$\forall x (R(x) \rightarrow C(x)) \wedge \forall x (P(x) \rightarrow \neg C(x)) \wedge \exists x (P(x) \wedge V(x)) \wedge \neg\exists x (V(x) \wedge \neg R(x))$

C_1. $\neg R(x) \vee C(x)$
C_2. $\neg P(y) \vee \neg C(y)$
C_3. $P(a)$
C_4. $V(a)$
C_5. $\neg V(z) \vee R(z)$

Aplicando Resolución vamos obteniendo las siguientes nuevas cláusulas:

C_6.	$\neg C(a)$	de 2 y 3 con { *a/y* }
C_7.	$\neg R(a)$	de 1 y 6 con { *a/x* }
C_8.	$\neg V(a)$	de 5 y 7 con { *a/z* }
C_9.	NADA	de 4 y 8

Árbol de refutación:



4. ESTRATEGIAS DE CONTROL PARA MÉTODOS DE RESOLUCIÓN

La regla de resolución nos indica cómo deducir una consecuencia a partir de dos cláusulas, pero no dice qué cláusulas escoger, ni que literales hay que unificar. Con motivo de encontrar qué cláusulas han de resolverse y qué resolución ha de realizarse con ellas, vamos a dar unas *estrategias de resolución*. Hay por tanto dos decisiones a tomar al aplicar la resolución:

1. seleccionar las cláusulas a unificar, y
2. seleccionar los átomos a unificar dentro de esas clásulas.

El punto 1 se conoce como *regla de búsqueda* y el punto 2 como *regla de selección*.

Es preciso aplicar criterios selectivos de manera que simplifiquen el proceso y así, poder abordarlo desde el punto de vista de la programación, evitando la generación de todas las combinaciones posibles, que al crecer el número de cláusulas de partida puede hacerse inmanejable. El estudio de estas estrategias pertenece al campo de la Inteligencia Artificial y aunque aquí hagamos un pequeño esbozo, están desarrollados más a fondo en libros que abarquen dicha materia.

Pero nos encontramos con dos problemas:

- Todas las estrategias de busqueda en la resolución están sujetos al *problema de la explosión exponencial*. Esta complejidad exponencial impide que puedan tener éxito pruebas que seguirían largas cadenas de inferencia. Las estrategias de búsqueda pueden reducir el exponente, pero no evitan su carácter exponencial.
- También están sujetas al *problema del paro*, por lo que no hay garantía de que la búsqueda termine, a menos que realmente exista una contradicción.

De ahí que, como ya hemos comentado repetidamente, se diga que dichos procedimientos son semidecidibles, ya que garantizan que nos van a decir si una fórmula es un teorema sólo cuando de hecho lo es.

Veamos ahora algunas estrategias de control y algunas estrategias de simplificación. Una *estrategia de control* es la que crea el árbol de refutación y por tanto define la obtención de nuevas clásulas. Se dice que es **completa** si su uso conduce a un procedimiento que llegará a encontrar una contradicción, si ésta existe. Las *estrategias de simplificación* nos permitirán reducir el número de cláusulas de la fórmula o el número de literales de las cláusulas, reduciendo por tanto el tamaño de las fórmulas con las que trabajaremos.

4.1. Estrategia a lo ANCHO

1º.- En el *nivel 0* del árbol están las cláusulas originales.

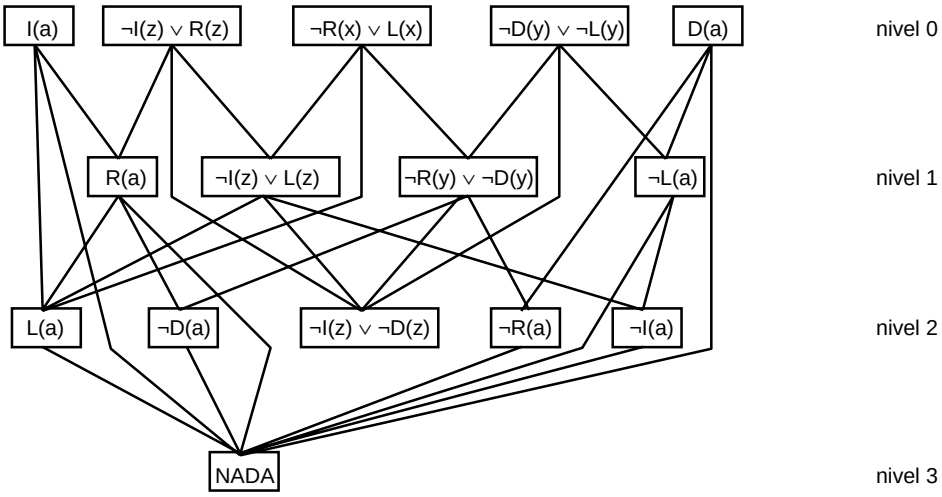
2º.- Se calculan todas las resolventes del *nivel 0* y se obtiene el *primer nivel*.

3º.- Obtener las resolventes del *2º nivel*. Y así sucesivamente (una resolvente de *nivel i-ésimo* es una cuyo padre de nivel más profundo es de *nivel (i-1)-ésimo*).

Ejemplo:

Cláusulas originales: I(a), ¬I(z) ∨ R(z), ¬R(x) ∨ L(x), ¬D(y) ∨ ¬L(y), D(a)

Construimos el *árbol de derivación*:



4.2. Estrategia del CONJUNTO SOPORTE

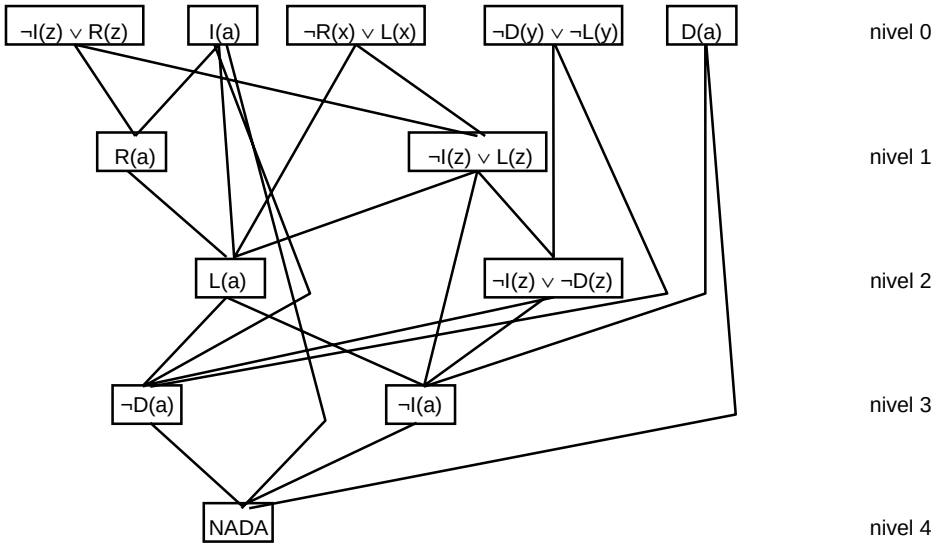
1º.- Obtener resolventes de cláusulas de forma que uno de los padres sea la cláusula negación de la fbf objetivo o alguna de las resolventes descendientes de ella (que constituyen el conjunto soporte). Con esto se consigue el *primer nivel* del árbol.

2º.- Las resolventes del 2º nivel se consiguen con las resolventes obtenidas en el *nivel 1* y con las cláusulas de los niveles anteriores.

3º.- Así sucesivamente hasta llegar a la resolvente *NADA*.

Ejemplo:

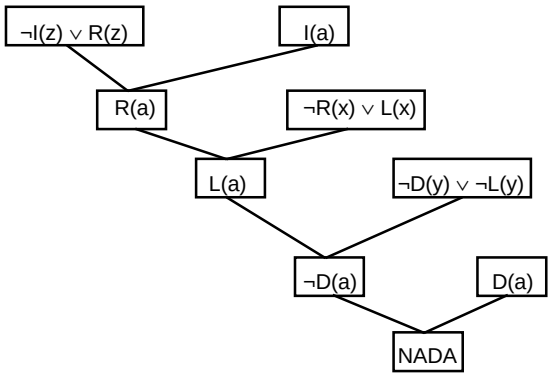
Seguimos con el mismo conjunto de cláusulas del ejemplo anterior y sabiendo que la negación de la fbf objetivo es *¬I(z) ∨ R(z)*. Luego para conseguir el *1º nivel* tenemos que trabajar sólo con ésta cláusula y obtener los resolventes que se puedan con las otras cláusulas.



4.3. Estrategia de PREFERENCIA DE UNIDADES

Es una modificación de la del conjunto soporte. En vez de completar a lo ancho, se *selecciona una cláusula con un sólo literal* (llamada unidad), como padre de la resolución. Cada vez que se usa una unidad en la resolución, la resolvente tiene menos literales que su otro padre. Este procedimiento se encamina hacia la cláusula vacía, y así incrementa la eficiencia.

Ejemplo:



4.4. ESTRATEGIAS DE SIMPLIFICACIÓN

Hay ocasiones en las que un conjunto de cláusulas puede simplificarse eliminando alguna de ellas o bien eliminando ciertos literales en las cláusulas. Estas simplificaciones deben conservar la incompatibilidad si el conjunto original lo era. Con la simplificación reducimos el crecimiento del conjunto de cláusulas que se están generando. Se pueden aplicar los siguientes criterios:

1.- Eliminación de Tautologías

Cualquier cláusula que contenga un literal y su negación (tautología) puede ser eliminada, pues cualquier conjunto incompatible que contenga una tautología seguirá siéndolo después de suprimirla, y recíprocamente.

Ejemplos:

$P(x) \vee B(y) \vee \neg B(y)$ se puede suprimir
$P(f(a)) \vee \neg P(f(a))$ se puede suprimir.

2.- Eliminación por subsunción

Por definición una cláusula $\{L_i\}$ *subsume* otra $\{M_i\}$ si existe una sustitución S tal que $\{L_i\}$ S es un subconjunto de $\{M_i\}$.

Ejemplos:

$P(x)$ subsume $P(y) \vee Q(z)$
$P(x)$ subsume $P(a)$
$P(x)$ subsume $P(a) \vee Q(z)$
$P(x) \vee Q(a)$ subsume $P(f(a)) \vee Q(a) \vee R(y)$

5. OBTENCIÓN DE RESPUESTAS MEDIANTE REFUTACIÓN POR RESOLUCIÓN

Vamos a introducir la idea de trabajo mediante un ejemplo. Supongamos que sabemos que “Pedro va a donde quiera que va Juan” y “Juan está en el colegio”. Ahora nos planteamos la pregunta “¿Donde está Pedro?”.

Formalización:

$\forall x \; [(E(juan, x) \rightarrow E(pedro, x))]$
$E(juan, colegio)$
$\exists x \; E(pedro, x)$

Idea: Para realizar la contestación de ésta pregunta la idea clave es convertir la pregunta en una fbf objetivo que contenga un cuantor existencial tal que la variable cuantificada existencialmente represente una respuesta a la pregunta.

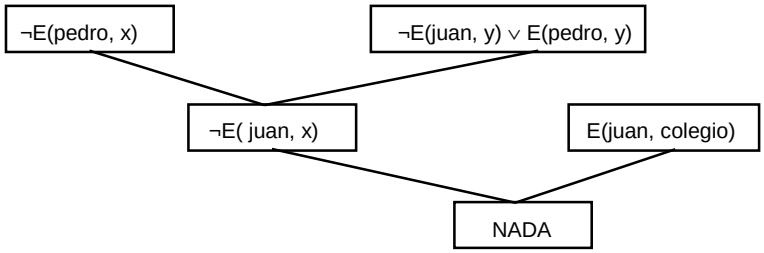
La refutación por resolución se obtiene negando la fbf que se desea probar, añadiendo esta negación al conjunto de fbf de partida, convirtiendo este conjunto en forma clausal y mostrando que este conjunto de cláusulas es incompatible. Las cláusulas que resultan de las fbf S se denominan *axiomas*.

Negamos la conclusión y obtenemos la Forma Clausal:

$\neg\exists x\ E(pedro, x) \Leftrightarrow \forall x\ \neg E(pedro, x) \Leftrightarrow \neg E(pedro, x)$

$\forall x\ [(\ E(juan, x) \rightarrow E(pedro, x)\ )\ ] \Leftrightarrow \neg E(juan, y) \vee E(pedro, y)$ (axioma)
$E(juan, colegio)$ (axioma)

Árbol de refutación:



A continuación extraemos una **respuesta** a la pregunta dada mediante ese mismo árbol de refutación, pero con la siguiente modificación.

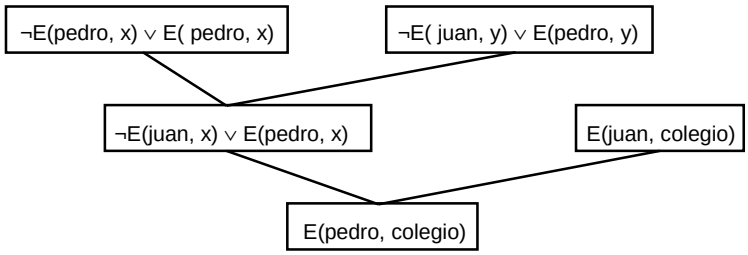
Proceso:

1.- Añadir a cada cláusula de las que origina la negación de la fbf objetivo, su propia negación. Así, *¬E(pedro, x)* da lugar a la tautología:

$\neg E(pedro, x) \vee E(pedro, x)$

2.- Siguiendo la estructura del árbol de refutación, realizar las mismas resoluciones que antes hasta encontrar una sola cláusula en la raíz.

3.- La cláusula raíz es una sentencia de respuesta.



La única diferencia con la pregunta es que en la respuesta *se ha sustituido la variable* que teníamos por una constante. La respuesta por tanto es "en el colegio".

Cuando hacemos un árbol de refutación, en la raíz obtenemos la cláusula vacía NADA y entonces sabemos que hay una incompatibilidad, o sea que el argumento es correcto. Obtenemos una respuesta si en lugar de esa cláusula encontramos una cláusula como respuesta. Como esta conversión se hace sustituyendo cada una de las cláusulas en que se convierte la negación de la fbf objetivo por una tautología, *el árbol de demostración* obtenido es una demostración por resolución de que la sentencia que figura en la raíz se sigue lógicamente de los axiomas más las tautologías. De ahí que también se sigue de los axiomas tan solo.

Ejemplo:

Premisa 1: Para todo *x* y para todo *z*, si existe un *y* del que *x* es padre y a su vez este *y* es padre de *z*, entonces *x* es abuelo de *z*

Premisa 2: Cualquier persona tiene padre

Pregunta: ¿ Existen individuos *x* e *y* tales que *x* es abuelo de *y* ?

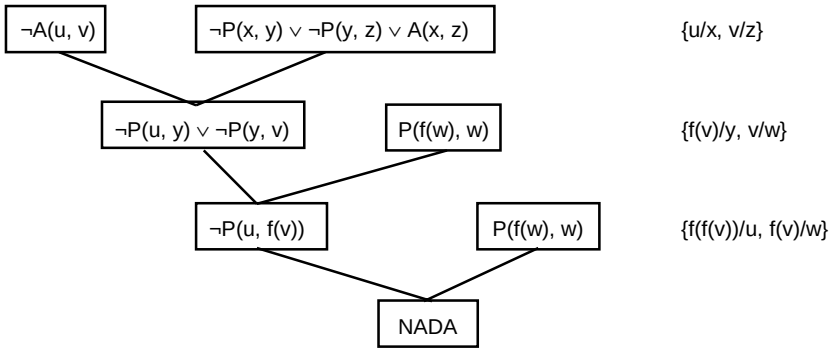
Formalización: $\forall x \, \forall z \; [\, \exists y \, (P(x, y) \wedge P(y, z)) \rightarrow A(x, z) \,]$
$\forall y \, \exists x \;\; P(x, y)$
$\exists x \, \exists y \;\; A(x, y)$

Demostramos esta fórmula por *refutación por resolución*:

Negación del objetivo: $\neg[\, \exists x \;\; \exists y \;\; A(x, y) \,] \iff \forall x \, \forall y \, \neg A(x, y)$

Obtención de las cláusulas:

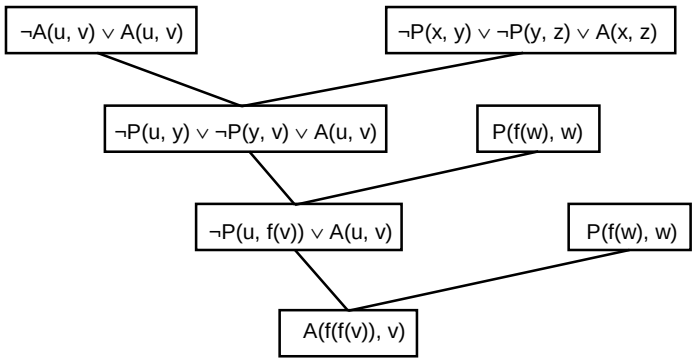
$\neg P(x, y) \vee \neg P(y, z) \vee A(x, z)$	axioma 1
$P(f(w), w)$	axioma 2
$\neg A(u, v)$	



NOTA: la función f(w) se utiliza para representar el padre de cada individuo.

El *árbol de demostración modificado* es el siguiente: la negación de la fbf objetivo se transforma en una tautología y las resoluciones se han realizado siguiendo las pautas de las del árbol anterior. Cada resolución del árbol modificado usa conjuntos de unificación que corresponden precisamente a los conjuntos de unificación del árbol de refutación.

Árbol de Demostración modificado:



Esta cláusula de la raíz representa: *∀v A(f(f(v)), v)* que es la sentencia respuesta. Esta proporciona una respuesta a la pregunta ¿Hay *x* e *y* tal que *x* sea abuelo de *y*?. La respuesta implica la definición de la función *f* (el padre de), y así, cualquier *v* y el padre del padre de *v* son ejemplos de individuos que satisfacen las condiciones de la pregunta.

En la práctica, la respuesta se puede obtener por la composición de las sustituciones realizadas en el proceso de unificación.

Ejemplo:
Si calculamos la composición de las sustituciones del ejemplo anterior:

s = {u/x, v/z} o {f(v/y, v/w_1} o {f(f(v))/u, f(v)/w_2} = {f(f(v))/x, v/z, f(v)/y, v/w_1, f(f(v))/u, f(v)/w_2}

y como la pregunta era ¬A(u, v) tenemos que la respuesta es:

[¬A(u, v)] s = ¬A(f(f(v)), v)

6. EJERCICIOS

1.- Calcular la composición $S_1 \circ S_2$ de las siguientes sustituciones:

S_1 = { f(x,y,a) / w , g(x) / z , h(a) / y } S_2 = { g(b) / x , b / w }

Aplicamos la sustitución S_2 a los términos de S_1:

{ f(g(b),y,a) / w , g(g(b)) / z , h(a) / y }

y añadimos los pares de S2 que tengan variables que no aparezcan en S1 como variables:

{ f(g(b),y,a) / w , g(g(b)) / z , h(a) / y , g(b) / x }

$S_1 \circ S_2$ = { f(g(b),y,a) / w , g(g(b)) / z , h(a) / y , g(b) / x }

2.- Encontrar, de existir, el UMG (unificador más general) de las siguientes expresiones:

a) **{ P(f(y), w, g(z)), P(u, u, v) }**

UMG = { f(y)/u, f(y)/w, g(z)/v }

b) **{ P(f(y), w, g(z)), P(v, u, v) }**

No tiene UMG, es decir , estas dos expresiones no pueden unificar ya que por la segunda expresión, el primer y el tercer argumento del predicado *P* deben ser iguales, y por la primera expresión tenemos *f(y)* y *g(z)* que no pueden unificar.

c) **{ P(a, x, f(g(y))), P(z, h(z,w), f(w)) }**

UMG = { a/z, h(a,g(y))/x, g(y)/w }

d) **{ P(x, y), Q(x, f(x)) }**

No existe UMG para estas expresiones ya que se trata de predicados diferentes.

e) **{ P(x, a), P(f(z), y), P(f(b), w) }**

UMG = { f(b)/x, b/z, a/y, a/w }

f) **{ P(x, a), P(y, z, b) }**

No existe UMG ya que la primera expresión tiene dos argumentos y la segunda tiene tres argumentos y por lo tanto no pueden unificar.

g) **{ P(x, a), P(f(x), y) }**

Estas expresiones no tienen UMG ya que los primeros argumentos, *x* y *f(x)*, no pueden unificar por contener la función a la variable.

3.- Realiza la traza del algoritmo para obtener el Unificador Más General (UMG) de las siguientes fórmulas:

P(f(x), g(x)) **P(y, g(a))**

UMG { [P, [f, x], [g, x]], [P, y, [g, a]] }
1. Ninguno es átomo
2. Z_1 := UMG { P, P }
   1.1. *P* y *P* son idénticos, devolver *NADA*
   $Z_1 \leftarrow$ *NADA*

Z_2 := UMG { [[f, x], [g, x]]NADA, [y, [g, a]]NADA } = UMG { [[f, x], [g, x]], [y, [g, a]] }
1. Ninguno es átomo
2. Z_1 := UMG { [f, x], y }
1.3. *y* símbolo de variable, devolver { *f(x)* / *y* }
$Z_1 \leftarrow$ { *f(x)* / *y* }
Z_2 := UMG { [g, x] { *f(x)* / *y* }, [g, a] { *f(x)* / *y* } = UMG { [g, x], [g, a] }
1. Ninguno es átomo
2. Z_1 := UMG { g, g }
1.1. *g* y *g* son idénticos, devolver *NADA*
$Z_1 \leftarrow$ *NADA*
Z_2 := UMG { [x]NADA, [a]NADA } = UMG { [x], [a] }
1.2. *x* símbolo de variable, devolver { *a* / *x* }
$Z_2 \leftarrow$ { *a* / *x* }
$\leftarrow$ *NADA* ° { *a* / *x* } = { *a* / *x* }
$Z_2 \leftarrow$ { *a* / *x* }
$\leftarrow$ { *f(x)* / *y* } ° { *a* / *x* } = { *f(a)* / *y* , *a* / *x* }
$Z_2 \leftarrow$ { *f(a)* / *y* , *a* / *x* }
$\leftarrow$ *NADA* ° { *f(a)* / *y* , *a* / *x* } = { *f(a)* / *y* , *a* / *x* }

UMG{ [P, [f, x], [g, x]], [P, y, [g, a]] } = { *f(a)* / *y* , *a* / *x* }

4.- Dado el siguiente argumento, demostrar su validez utilizando refutación por resolución:

$$\forall x\, \exists y\, [\, \neg A(x, y) \vee (B(x) \wedge C(y))\,]$$
$$\forall x\, [\, B(x) \rightarrow \neg \forall x\, \forall y\, D(x, y)\,]$$
$$\neg \exists x\, E(x)$$
$$\forall x\, \forall y\, D(x, y) \wedge \exists x\, E(x) \quad \Rightarrow \quad \neg \forall x\, \forall y\, A(x, y)$$

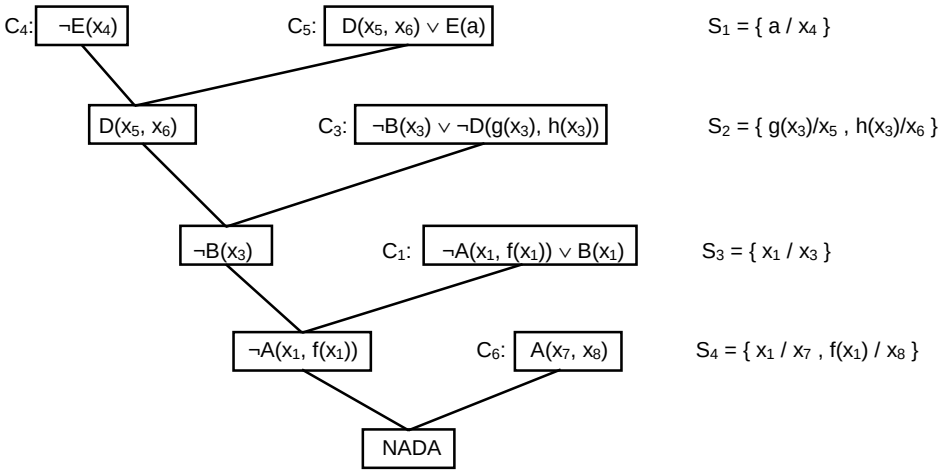
La forma clausal de las premisas ya la hemos obtenido en el ejemplo de la Forma Clausal (pag. 150). Por tanto nos queda negar la conclusión y transformarla a forma clausal:

$\neg\neg \forall x\, \forall y\, A(x, y)$
$\forall x\, \forall y\, A(x, y)$
$A(x, y)$

Así, obtenemos las siguientes cláusulas, después de normalizar variables:

C_1: $\neg A(x_1, f(x_1)) \vee B(x_1)$
C_2: $\neg A(x_2, f(x_2)) \vee C(f(x_2))$
C_3: $\neg B(x_3) \vee \neg D(g(x_3), h(x_3))$
C_4: $\neg E(x_4)$
C_5: $D(x_5, x_6) \vee E(a)$
C_6: $A(x_7, x_8)$

El árbol de refutación con las sustituciones quedaría:



7. TRABAJOS COMPLEMENTARIOS

Estudio de las diferentes *estrategias de resolución*, tanto en la elección del conjunto de cláusulas, como en los átomos a unificar.

8. LA LÓGICA EN LA VIDA

Una *paradoja* es una idea extraña, una expresión lógica en la que hay una incompatibilidad aparente. Los esfuerzos por resolver las paradojas han hecho avanzar a la lógica y a la ciencia en general. El término paradoja puede tener un amplio sentido :

- afirmaciones que parecen falsas, aunque en realidad son verdaderas
- afirmaciones que parecen verdaderas, pero que en realidad son falsas
- razonamientos aparentemente imposibles, y que sin embargo conducen a contradicciones lógicas
- declaraciones cuya veracidad o falsedad es indecidible.

1. En la puerta de una barbería había un cartel que decía:

> *Yo afeito a quienes no se afeitan a sí mismo,*
> *y nunca a nadie que se afeite a sí mismo*

¿ Quién afeita al barbero ?

2. Un anuncio de una revista dice:

> *¿ Quiere usted aprender a leer y escribir ?*
> *Aprenda rápidamente por correspondencia*
> *escribiéndonos a la dirección adjunta*

3. ¿ Es *No* la respuesta correcta a esta pregunta ?

4. Un río que separaba dos países estaba cruzado por un puente en cuyo extremo había un guardián y una horca. Allí estaba colgado un cartel que decía: “Si alguno pasare por este puente de una parte a otra, ha de jurar primero adónde y a qué va; si jurare verdad, podrá pasar; y si dijere mentira, muera ahorcado”. En cierta ocasión, pasó un hombre que juró que iba para morir en la horca que allí había. ¿ De ser tú el guardián qué harías, dejarle pasar o ahorcarlo ?

Capítulo **9**

PROGRAMACIÓN LÓGICA

Llegados a este punto, hemos visto como nos puede servir la Lógica para representar el conocimiento declarativo sobre diferentes campos del saber, así como para extraer nuevos conocimientos a partir de los que poseamos mediante la demostración automática de teoremas. Toda esta base teórica dio como resultado la aparición de una clase de lenguajes de programación: la programación lógica. Como punto final vamos a ver como la Lógica de Primer Orden, o mejor dicho, un subconjunto de ella llamado cláusulas de Horn, puede dar lugar a los programas lógicos, y con ello, a un nuevo estilo de programación.

Una *representación declarativa* del conocimiento es aquella en la que dicho conocimiento está especificado, pero sin embargo, no viene dada la manera en que debe ser usado tal conocimiento. Por tanto, para utilizar el conocimiento de una representación declarativa debe disponerse de un mecanismo que especifique qué debe hacerse con el conocimiento y de qué modo debe hacerse. Por contra, una *representación procedimental* es aquella en la que la información del control necesaria para utilizar el conocimiento se encuentra embebida en el propio conocimiento.

Con la aparición del lenguaje LISP en los años sesenta surge un nuevo estilo de programación, los lenguajes de *programación declarativa*, basados en un formalismo abstracto (la teoría matemática del lambda calculus de Church en el caso de LISP y el cálculo de predicados en el caso de la programación lógica) que nos indica cómo usar el conocimiento dado en el programa. Un programa declarativo es aquel cuyas sentencias

tienen una interpretación declarativa, y por tanto pueden ser leídos como una descripción formal del problema sin tener que recurrir al comportamiento de ninguna máquina. Este modelo de lenguajes está claramente influido por un entendimiento matemático de las *descripciones*. El acercamiento hacia un concepto declarativo de la programación intenta aproximar las nociones de programa y de especificación. Las principales características que presentan este tipo de lenguajes son:

- Expresivos: las descripciones de los problemas no son difíciles de escribir.
- Fiables: protegen al usuario, en la medida de lo posible, de cometer errores.
- Matemáticamente elegantes: tienen un soporte matemático en la mayoría de actividades de la programación.

Aunque los orígenes de la programación declarativa datan de los primeros tiempos de la computación, sólo en los últimos años han alcanzado cierta aceptación. Ello puede ser atribuido por una parte al alcance de las inversiones en los lenguajes existentes, y por otra al "hueco semántico" entre los lenguajes declarativos y los ordenadores convencionales. Los avances tecnológicos recientes han estrechado este hueco y, por ello han aumentado las ventajas de la programación declarativa. Otro importante aspecto en favor de la programación declarativa es el desarrollo de las nuevas arquitecturas paralelas de los ordenadores.

Hay dos clases de lenguajes declarativos: los lenguajes funcionales y los lógicos. Nosotros trataremos los lenguajes lógicos. La programación lógica trata con relaciones en lugar de con funciones, lo que nos proporciona mayor flexibilidad, ya que las relaciones no tienen sentido de la dirección y tratan uniformemente argumentos y resultados (no hay distinción entre parámetros de entrada y de salida). Consideraremos las sentencias lógicas (fórmulas bien formadas) como representaciones del conocimiento, y por ello, como líneas de código de nuestros programas.

Cuando tenemos una colección de proposiciones, podemos investigar si se deriva algo interesante de ellas. Aquellas proposiciones que tomemos como verdaderas las llamaremos **axiomas** o **hipótesis**, y las proposiciones que encontremos que se deriven de ellas serán los **teoremas**. La actividad de deducir consecuencias interesantes a partir de proposiciones dadas la llamamos **comprobación de teoremas**.

1. LA PROGRAMACIÓN LÓGICA

La aparición de este nuevo paradigma de programación viene precedida de distintos estudios e investigaciones en el campo de la informática. Vamos a comentar brevemente algunos de ellos:

- El cambio cualitativo producido a primeros de los años setenta en el concepto de la programación introduciendo la metodología de diseño estructurado y la verificación de programas fue un primer paso de replanteamiento del diseño

de los procesos: en lugar de basados en *cómo computarlos*, basados en *cómo entenderlos*.

- Los desarrollos en el área de la deducción automática dieron lugar al inicio de una línea en la que se plantea la simple especificación lógica de los procesos prescindiendo de la especificación de procedimientos.
- La base de estos métodos es la regla de resolución de Robinson (1965) y las técnicas de extracción de respuestas iniciadas por Green (1969) y seguidas por Colmerauer (1972) y Kowalski (1974).

Así, la **Programación Lógica** se ha convertido en nueva línea de desarrollo de software, válida tanto para la especificación de algoritmos como para modelos de Inteligencia Artificial, ambivalencia que ha dado lugar a su utilización como base del proyecto de ordenadores de quinta generación (máquinas especializadas en el procesamiento de conocimientos).

La idea central de todo esto la podemos expresar utilizando la conocida ecuación de Kowalski

algoritmo = lógica + control

de manera que el *control* (estrategia para encontrar la solución) la dejamos en manos de la maquina, y el programador sólo debe preocuparse de la *lógica* (información acerca del problema que queremos resolver).

La Programación Lógica usa como base sentencias de la Lógica de Primer Orden, y trata de representar conocimiento mediante relaciones (predicados) entre objetos, de manera que un programa lógico estará constituido por un conjunto de relaciones, y su ejecución vendrá a demostrar que una nueva relación se sigue de las que constituían el programa. Aunque se asocia Programación Lógica con Prolog, no es lo mismo, y el lenguaje Prolog sólo es un caso particular, existiendo distintos sistemas basados en otras lógicas o que incorporan nuevas características: MOLOG (Lógica Modal), GOEDEL (Lógica “many-sorted”), PARLOG (Programación Lógica Paralela), Prolog++ (Programación Lógica Orientada a Objetos), CLP (Programación Lógica con restricciones), ...

Para finalizar con este apartado y conectar con los siguientes, sabemos que un lenguaje de programación queda totalmente definido por su *sintaxis* y su *semántica*. La sintaxis nos dirá “cómo” escribir nuestros programas, y la semántica “qué” significan dichos programas:

- La **sintaxis** es lo primero que tenemos que conocer de un lenguaje de programación para poder empezar a usarlo, y nos dice cuál es la forma correcta de escribir los programas en dicho lenguaje. El aspecto sintáctico de los programas lógicos lo veremos, de manera general, en el siguiente apartado

"Representación de programas lógicos" y en el anexo "Prolog" se dará la sintaxis concreta de este lenguaje.

- La **semántica** del lenguaje especifica el significado de los programas que podemos construir con este lenguaje. El conocimiento de la semántica será necesario para poder escribir programas correctos y a la vez ser capaces de predecir los efectos de la ejecución de cualquier instrucción.

2. REPRESENTACIÓN DE PROGRAMAS LÓGICOS

Como ya hemos visto, cualquier fórmula del Cálculo de Predicados puede ser transformada a **Forma Clausal**, que consiste en una colección de cláusulas, donde cada una de ellas es a su vez una disyunción de literales { L_1, ..., L_k}. Estos literales pueden ser fórmulas atómicas negadas o no. Así tenemos un subconjunto {A_1, ..., A_n} de literales afirmados y otro {$\neg N_1$, ..., $\neg N_m$} de negados.

$$\{L_1 , ..., L_k\} = \{A_1 , ..., A_n\} \cup \{\neg N_1 , ..., \neg N_m\}$$

La notación en Forma Clausal tiene la ventaja de que reduce a una forma única lo que se puede escribir de diversas formas. Esto resulta imprescindible si queremos llevar a cabo manipulaciones formales sobre fórmulas del Cálculo de Predicados y desarrollar procesos de automatización. De ahí la importancia de la forma clausal cuando tratamos de aplicar la lógica a la informática, y en concreto a la programación lógica.

2.1. Notación para la Programación Lógica

Para acercarnos en lo posible a la sintaxis de los lenguajes de programación, a partir de ahora, para las fórmulas lógica en forma clausal utilizaremos la siguiente notación específica de la Programación Lógica :

1. Como es una colección (conjunción) de cláusulas, las escribiremos en secuencia. Recordemos que el orden es irrelevante por tratarse de conjunciones, ya que estas cumplen las propiedades conmutativa y asociativa.

2. Cada cláusula es una colección (disyunción) de literales, que son fórmulas atómicas afirmadas o negadas:

- Escribiremos primero las fórmulas atómicas no negadas llamadas *cabeza de la cláusula*, seguidas de las negadas, *cuerpo de la cláusula*, separados ambos grupos por el símbolo especial "←" (si).

- Las fórmulas atómicas no negadas (cabeza de la cláusula) irán separadas por el símbolo ";" (o), y las negadas (cuerpo de la cláusula) serán escritas sin el negador (¬) y separadas por el símbolo "," (y).

Ejemplo 1:

Supongamos que tenemos la siguiente cláusula:

$$p \vee \neg q \vee \neg r \vee s \vee \neg t \vee u$$

si agrupamos y realizamos transformaciones:

$$\neg q \vee \neg r \vee \neg t \vee p \vee s \vee u$$

$$\neg(q \wedge r \wedge t) \vee (p \vee s \vee u)$$

$$(q \wedge r \wedge t) \longrightarrow (p \vee s \vee u)$$

que significa que:

"Si ocurren *q*, *r* y *t* entonces deben ocurrir *p*, *s* o *u*"

o lo que es lo mismo:

"Para resolver el problema *p*, *s* o *u*, antes deben resolverse *q*, *r* y *t*"

La cláusula anterior, escrita en la notación que acabamos de comentar quedaría:

$$p ; s ; u \leftarrow q , r , t$$

Ejemplo 2:

Tenemos la fórmula del Cálculo de Predicados:

$$\forall y \, \forall x (mujer(x) \wedge mujer(y) \wedge padresDe(x, p, m) \wedge padresDe(y, p, m) \longrightarrow hermanas(x, y))$$

1. Obtenemos la Forma Clausal correspondiente:

$$\forall y \, \forall x (\neg(mujer(x) \wedge mujer(y) \wedge padresDe(x, p, m) \wedge padresDe(y, p, m)) \vee hermanas(x, y))$$

$$\forall y \, \forall x (\neg mujer(x) \vee \neg mujer(y) \vee \neg padresDe(x, p, m) \vee \neg padresDe(y, p, m) \vee hermanas(x, y))$$

$$\neg mujer(x) \vee \neg mujer(y) \vee \neg padresDe(x, p, m) \vee \neg padresDe(y, p, m) \vee hermanas(x, y)$$

2. La cláusula la escribimos en la notación vista antes:
   hermanas(x, y) ← mujer(x), mujer(y), padresDe(x, p, m), padresDe(y, p, m).

que se puede leer como:

"Dos personas son hermanas si ambas son mujeres y tienen los mismos padres"

2.2. Cláusulas de Horn Definidas

Vamos a restringir el conjunto de fórmulas que utilizaremos a las llamadas *fórmulas de Horn*[1], lo cuál nos permitirá una mejor y más eficiente implementación de la comprobación de teoremas. De esta manera, a partir de ahora trabajaremos con una "sublógica" de la Lógica de Predicados, que llamaremos **programas lógicos**.

Cláusula de Horn Definidas : es una cláusula con, como máximo, un literal no negado.

Siguiendo con la representación utilizada antes para una cláusula:

$$\{L_1 , ..., L_k\} = \{A_1 , ..., A_n\} \cup \{\neg N_1 , ..., \neg N_m\}$$

diremos que es una cláusula de Horn si $n \leq 1$. Así podemos diferenciar los siguientes tipos de cláusulas de Horn:

1.- Hipótesis o cláusulas con un literal no negado (n=1), llamadas *cláusulas con cabeza*, que pueden ser a su vez:

- **Hechos**: sin literales negados (m=0) $\{A\}$

Ejemplo: mujer(lirios) ←

- **Reglas**: con literales negados (m>0) $\{A,\neg N_1,...,\neg N_m\}$

Ejemplo: hermanas(x, y) ← mujer(x),
mujer(y),
padresDe(x, p, m),
padresDe(y, p, m)

2.- Preguntas o cláusulas con ningún literal no negado (n=0) o teoremas a demostrar, llamadas *cláusulas sin cabeza* o decapitadas

- **Metas** u **objetivos**: (m>0) $\{\neg N_1 , ..., \neg N_m\}$

Ejemplo: ← hermanas(lirios, rosana)

[1] Reciben este nombre por que este tipo de fórmulas fueron investigadas por primera vez por el lógico Alfred Horn (1951).

Todo problema resoluble con cláusulas generales, tiene un modelo equivalente resoluble en Cláusulas de Horn, por lo que las restricciones de notación no conllevan una pérdida de capacidad de representación, y en cambio, sí que facilitan la formulación.

3. SEMÁNTICA DE PROGRAMAS LÓGICOS

Como ya hemos dicho antes, la semántica de un lenguaje especifica el significado de los programas que podemos construir con dicho lenguaje. Para hacer la definición formal de la semántica de un lenguaje existen distintos métodos. Vamos a comentar brevemente algunos de ellos.

- La *semántica declarativa* es un tipo de semántica que especifica el significado de los objetos sintácticos por medio de su traducción en elementos y estructuras de un dominio matemático conocido :
  - La semántica denotacional trata al programa como si fuera un conjunto de funciones matemáticas, la composición de las cuales nos dará su significado. Las funciones se definen en términos de cambios de estado ; dado un constructor y un estado, obtenemos un nuevo estado.
  - La semántica por teoría de modelos expresa el significado de un programa por el conjunto de sus consecuencias lógicas. Se basa en los conceptos lógicos de interpretación y modelo.
- La *semántica operacional* es la más cercana a la intuición del programador. En un primer momento define una máquina abstracta que soporte un conjunto de operaciones y estructuras de datos simples ; posteriormente se definen los constructores del lenguaje en términos de la máquina definida. La semántica operacional sería como un interprete de nuestro lenguaje en la máquina abstracta, así el significado de nuestro programa se define en términos de las acciones que serían ejecutadas por dicho modelo abstracto de máquina.

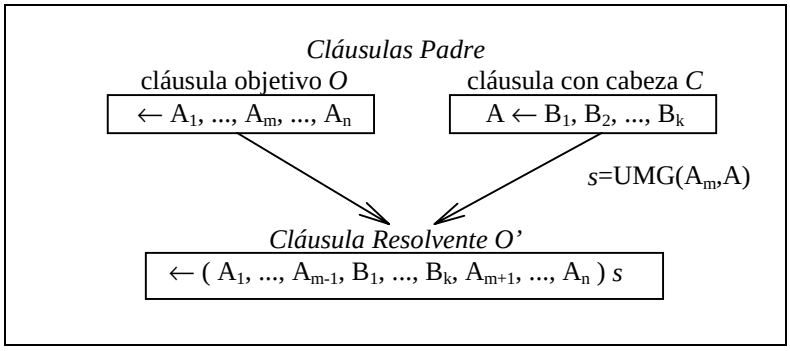
Vamos a estudiar la semántica operacional (la regla de resolución) y la semántica declarativa (modelo de Herbrand mínimo) de los programas lógicos.

3.1. Semántica Operacional

El principio de resolución de Robinson, como hemos visto en el tema anterior, nos dice de qué manera puede derivarse una proposición a partir de otras. Así, utilizando la técnica de refutación podemos desarrollar un método de demostración

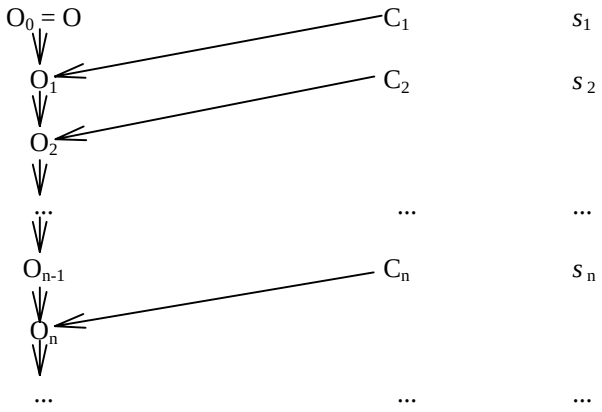
puramente mecánico ya que una importante propiedad formal que presenta la resolución es la de ser de *refutación completa*, es decir, si un conjunto de cláusulas no es *coherente*, la resolución podrá deducir de ellas la cláusula vacía. Así, si nuestras hipótesis son coherentes, sólo tenemos que añadirles la negación de lo que queremos probar. Si por resolución deducimos la cláusula vacía, nuestro objetivo quedará probado.

El problema es que la resolución no nos dice cómo decidir qué cláusulas mirar ni qué literales unificar. Para ello están las *estrategias de resolución*, que caen dentro del campo de la Inteligencia Artificial, aunque en el tema anterior hayamos esbozado alguna de ellas. La *Resolución Lineal* es una estrategia sencilla que nos dice que debemos resolver cláusulas con cabeza con cláusulas objetivo (sin cabeza).



Resolución Lineal

Dado un conjunto de cláusulas definidas (programa lógico) y una cláusula negativa (objetivo), una *derivación lineal* es :



donde

$O_0, O_1, ..., O_n, ...$	es una secuencia de objetivos
$C_1, C_2, ..., C_n, ...$	es una secuencia de cláusulas del programa *P*
$s_1, s_2, ..., s_n, ...$	es una secuencia de unificadores más generales para O_{i-1} y C_i, respectivamente.

Al utilizar resolución lineal el programa lógico estará razonando hacia atrás desde el objetivo propuesto hasta que encuentre el modo de terminar utilizando las cláusulas del programa, es decir se parte de las metas a conseguir. Este método de razonamiento se denomina también *razonamiento dirigido al objetivo*. Esto resultará adecuado para sistemas interrogadores. Por contra, también existe el razonamiento hacia adelante, que parte de las cláusulas iniciales. Esta elección de una estrategia fija (SLD-Resolución) en el control de la búsqueda representa una desventaja frente a otros tipos de programación (LISP, ...). Llamamos SLD-Resolución porque:

- utilizamos cláusulas de Horn **D**efinidas
- utilizamos la estrategia de búsqueda **L**ineal: primero en profundidad
- utilizamos una regla de **S**elección: primero a la izquierda

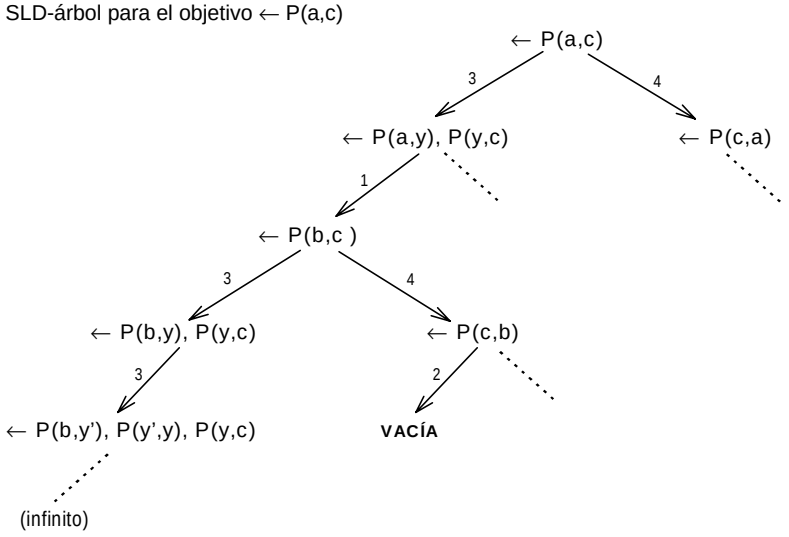
De igual forma, podemos hablar de SLD-Refutación. A partir de un conjunto de cláusulas *C* y un objetivo a cumplir *O* tendremos una SLD-Refutación si obtenemos una SLD-Derivación finita que tiene la cláusula vacía como el último objetivo de la derivación.

Ejemplo :

Dado el siguiente programa lógico *P* :

C_1 : P(a,b) ←
C_2 : P(c,b) ←
C_3 : P(x,z) ← P(x,y), P(y,z)
C_4 : P(x,y) ← P(y,x)

obtener el SLD-árbol para el objetivo ← P(a,c)



Como podemos observar en el ejercicio anterior, si la estrategia utilizada para la regla de búsqueda es el *primero en profundidad* y para la regla de selección el *átomo más a la izquierda*, la anterior derivación seria infinita, mientras que vemos claramente que existe una derivación que nos lleva a la cláusula vacía en pocos pasos.

Todo problema representado en cláusulas de Horn será resoluble si tenemos sólo una cláusula decapitada, y el resto con cabeza, es decir, si sólo existe una pregunta:

- *Al menos una cláusula decapitada*: al aplicar el método de resolución a dos cláusulas de Horn con cabeza, obtenemos como resultado otra cláusula con cabeza, con lo que no podríamos derivar la cláusula vacía.
- *Sólo una cláusula decapitada*: de existir varias cláusulas decapitadas, cualquier prueba por resolución de una nueva cláusula puede ser convertida en una prueba usando como mucho una de ellas. Al resolver con la primera cláusula decapitada, obtenemos una nueva cláusula decapitada, y por tanto no necesitamos resolver con las otras cláusulas decapitadas.

Al programar en lógica, los programas son nuestras hipótesis acerca del Universo, y las preguntas son como teoremas que se pretende demostrar. Una máquina lógica pone en marcha los mecanismos de inferencia lógica para buscar las respuestas a cualquier pregunta planteada. Si el programa contiene suficientes conocimientos para que puedan deducirse soluciones, estas le serán dadas al usuario.

Sea P un Programa Lógico y $\leftarrow N_1 , \ldots , N_m$ un objetivo no vacío, demostrar que $P \cup \{\leftarrow N_1 , \ldots , N_m \}$ no posee un modelo (es insatisfacible) es equivalente a demostrar que $\exists (N_1 \wedge \ldots \wedge N_m)$ se sigue de P .

3.2. Semántica Declarativa

Para realizar el estudio de la semántica declarativa caracterizaremos a un programa lógico *P* con el Modelo Mínimo de Herbrand. Para ello nos basaremos en los conceptos vistos en el capítulo anterior: universo de Herbrand, base de Herbrand, interpretación de Herbrand, ...

Por tanto hablaremos de *Modelo de Herbrand* (*M*) de un programa lógico para denotar a una interpretación de Herbrand que es modelo, es decir que hace ciertas todas las cláusulas de dicho programa.

Podemos ver el significado de un programa lógico *P* como el dado por un modelo de Herbrand de *P*. Pero algunos modelos son más grandes que lo estrictamente necesario. Por ello podemos tomar la intersección de todos los modelos de Herbrand,

que, evidentemente, también será un modelo para *P* y no existirá ningún modelo más pequeño.

Modelo de Herbrand Mínimo (M_P) : intersección de todos los modelos de Herbrand para ese programa

$$M_P = \bigcap_{i=1}^{\infty} M_i$$

Se puede demostrar que M_P es el conjunto de todos los objetivos básicos (sin variables) que se pueden obtener por resolución usando las cláusulas del programa *P*, y por tanto de todas las fórmulas atómicas que son consecuencia lógica de *P*.

Algunas veces el modelo mínimo de Herbrand no se puede obtener fácilmente a simple vista. Veamos un método recursivo para calcularlo. Para ello utilizaremos el operador de consecuencias inmediatas, definido de la siguiente manera

$$f_P(M) = M \cup \{ A \,/\, A \leftarrow B_1, ..., B_n \ ^* \text{ y } \{B_1, ..., B_n\} \subseteq M \}$$

* $A \leftarrow B_1, ..., B_n$ es una instancia básica (sin variables) de una cláusula del programa *P*

El modelo mínimo de Herbrand será el menor punto fijo de f_P, es decir la menor solución de la ecuación $f_P(M) = M$, por lo que podemos partir del conjunto vacío y parar cuando no sea modificado.

Ejemplo :

Sea el siguiente programa lógico

P(a) ←
Q(b) ←
R(c) ←
P(x) ← Q(x)
R(y) ← P(y)

El universo de Herbrand será : H = { a, b, c }
La base de Herbrand será : B = { P(a), P(b), P(c), Q(a), Q(b), Q(c), R(a), R(b), R(c) }
Un modelo de Herbrand será : M = { P(a), P(b), Q(a), Q(b), R(a), R(b), R(c) }

En cambio no lo será : M' = { P(a), P(b), Q(a), Q(b), R(a), R(c) }

El modelo de Herbrand mínimo lo calculamos, de manera recursiva :

$M_0 = \{\}$
$M_1 = f_P(M_0) = \{ P(a), Q(b), R(c) \}$
$M_2 = f_P(M_1) = \{ P(a), Q(b), R(c), P(b), R(a) \}$
$M_3 = f_P(M_2) = \{ P(a), Q(b), R(c), P(b), R(a), R(b) \}$
$M_4 = f_P(M_3) = \{ P(a), Q(b), R(c), P(b), R(a), R(b) \} = M_3$

M_P = { P(a), Q(b), R(c), P(b), R(a), R(b) }

4. PROLOG

El lenguaje Prolog (PROgramación LOGica), definido por Colmerauer en 1972, es el mejor representante de esta nueva concepción de la programación. Los conceptos básicos de este lenguaje, así como algunos ejemplos, los podemos encontrar en el anexo, por lo que aquí no vamos a extendernos en detalles.

Un sistema Prolog está basado en un comprobador de teoremas por Resolución usando cláusulas de Horn, y la estrategia de «Backtracking» (resolución de entrada lineal). La Base de Conocimientos de Prolog estará formada por cláusulas positivas o cláusulas con cabeza (*hechos* y *reglas*). Su ejecución consistirá en la introducción de una cláusula negada u *objetivo* que queremos hacer cumplir.

Ejemplo 1:

(población en miles de habitantes y superficie en miles de km.^{2})

Hechos:

```
poblacion(alicante,1146).
poblacion(castellon,434).
poblacion(valencia,2068).

superficie(alicante,6).
superficie(castellon,7).
superficie(valencia,11).
```

Reglas:

```
densidad(X, Y) :-     poblacion(X, P),
                      superficie(X, S),
                      Y is P/S.
```

Pregunta:

```
?- densidad(alicante, X).

X=191
```

Pregunta:

```
?- densidad(X,Y).

X=alicante
Y=191
More (y/n)? y

X=castellon
Y=62
More (y/n)? y

X=valencia
Y=188
More (y/n)? y

no
```

Ejemplo 2:

Programa:

```
/* factorial(N,F) <- F es el factorial de N */
factorial(0,1).
factorial(N,F) :- N1 is N-1,
                  factorial(N1,F1),
                  F is N*F1.
```

Pregunta:

```
?- factorial(4,24).
Yes
```

Pregunta:

```
?- factorial(4,F).
F=24
```

Como resumen final, veamos un cuadro comparativo de las diferentes notaciones usadas para las fórmulas bien formadas del Cálculo de Predicados :

	Fórmula	**Cláusulas de Horn**	**Program. Lógica**	**Prolog**
REGLA	$\forall (A \vee \neg N_1 \vee ... \vee \neg N_m)$ $\forall ((N_1 \wedge ... \wedge N_m) \rightarrow A)$	$\{A, \neg N_1 , ... , \neg N_m \}$	$A \leftarrow N_1 , ... , N_m$	A :- N_1 , ..., N_m .
HECHO	$\forall (A)$	{ A }	$A \leftarrow$	A.
OBJETIVO	$\forall (\neg N_1 \vee ... \vee \neg N_m)$ $\neg \exists (N_1 \wedge ... \wedge N_m)$	$\{ \neg N_1 , ... , \neg N_m \}$	$\leftarrow N_1 , ... , N_m$	?- N_1 , ... , N_m .

5. TRABAJOS COMPLEMENTARIOS

1.- Aplicaciones de la Programación Lógica.

2.- Implementación en Prolog de los pasos para la obtención de la Forma Clausal de una fórmula bien formada del Cálculo de Predicados.

3.- Estudio de distintas clases de programación lógica: CLP, ...

6. LA LÓGICA EN LA VIDA

1. En un cuento de Gordon Dickson, “The Monkey Wrench” unos científicos consiguen salvar la vida inutilizando un ordenador. La técnica que emplearon fue decirle a la máquina: “Tienes que rechazar el enunciado que te estoy proponiendo, porque todos los enunciados que yo propongo son incorrectos”.

2. A uno de los primeros programas de traducción entre el ruso y el inglés se le introdujo la frase “el espíritu está presto, pero la carne es débil”. La tradujo al ruso y luego al inglés de nuevo, siendo el resultado “el vodka es agradable, pero la carne está demasiado blanda”.

3. ***Test de Turing*** [2] para determinar razonablemente si una máquina piensa: un ordenador y un humano se ocultan a la vista del interrogador; este debe de tratar de averiguar cuál es el ordenador y cuál es el ser humano mediante el planteamiento de preguntas a cada uno de ellos; el humano responderá a las preguntas sinceramente tratando de persuadirle de que él es realmente el ser humano y el otro es el ordenador, y la computadora está programada para mentir intentando convencer al interrogador de que es el ser humano y el otro la máquina. Si el interrogador es incapaz de identificar de una forma definitiva al ser humano real, el ordenador ha superado la prueba.

4. ***La Sala China*** [3]: supongamos que tenemos un sistema que pasa la prueba de Turing. Dicho sistema podría semejarse a un sistema formado por un humano que únicamente entiende el castellano encerrado en una habitación con un libro de reglas. Mediante una apertura recibe del exterior unas papeletas con simbolos indescifrables, pero que localizados en el libro de reglas le permiten ejecutar una serie de instrucciones. La salida final es devuelta al exterior. Visto desde el exterior este sistema trabaja con oraciones escritas en chino como entrada y produce una respuesta también en chino. Pero, si el humano no entiende el chino, el libro de reglas no entiende chino ¿aquí nadie comprende el chino?

5. “Si el cerebro del ser humano fuera tan sencillo que lo pudiéramos entender, entoncces seríamos tan estúpidos que tampoco lo entenderíamos”

   Jostein Gaarder, “El mundo de Sofía”

[2] Descrito por primera vez en el artículo “Computing Machinery and Intelligence” de Alan M. Turing publicado en 1950 en la revista “Mind”. Reimpreso en "Computation & Intelligence", Collected Reading editados por George F. Luger, AAAI Press y The MIT Press, 1995

[3] Descrito por primera vez en el artículo “Minds, Brains, and Programs” de John R. Searle publicado en 1980 en la revista “The Behaviorial and Brain Sciences” 3, pág. 417-424.

Anexo **A**

PROLOG

En este anexo hacemos una breve exposición del lenguaje de programación lógica más conocido: PROLOG. Con ello pretendemos presentar una aplicación directa a la Informática de los conceptos y mecanismos de la Lógica de Primer Orden desarrollados a lo largo de los temas teóricos. La base de Prolog es la lógica clausal, y más concretamente las cláusulas de Horn, y su forma de ejecución es el principio de resolución. La Programación Lógica trata con relaciones entre objetos. Las relaciones serán expresadas en forma de hechos y reglas. La ejecución de un programa consiste en demostrar que una conclusión se deduce de nuestra base de conocimientos realizando preguntas. Además, Prolog ha conseguido incrementar su eficiencia introduciendo predicados extralógicos.

El lenguaje de programación PROLOG ("PROgrammation en LOGique") fue creado por Alain Colmerauer y sus colaboradores alrededor de 1970 en la Universidad de Aix-Marseille[1], si bien uno de los principales protagonistas de su desarrollo y promoción fue Robert Kowalski[2] de la Universidad de Edimburgh. Las investigaciones de Kowalski proporcionaron el marco teórico, mientras que los trabajos de Colmerauer dieron origen al actual lenguaje de programación, construyendo el primer interprete Prolog. David Warren[3], de la Universidad de Edimburgh, desarrolló el primer compilador de

[1] A. Colmerauer. *Les Systèmes-Q, ou un formalisme pouranalyser et synthétiser des phrases sur ordinateur*. Internal Report 43, Computer Science Dpt., Université de Montreal, septiembre, 1970.
A. Colmerauer, H. Kanoui, P. Roussel y R. Pasero. *Un Systeme de Communication Homme-Machine en Français*, Groupe de Recherche en Intelligence Artificielle, Université d'Aix-Marseille, 1973.

[2] R. Kowalski. *Predicate Logic as a Programming Language*. En Proc. IFIP, Amsterdam, 1974.

[3] D. Warren. *The runtime environment for a prolog compiler using a copy algorithm*. Technical Report 83/052, SUNY and Stone Brook, New York, 1983.

Prolog (WAM – "Warren Abstract Machine"). Se pretendía usar la lógica formal como base para un lenguaje de programación, es decir, era un primer intento de diseñar un lenguaje de programación que posibilitara al programador especificar sus problemas en lógica. Lo que lo diferencia de los demás lenguajes de programación es el énfasis sobre la especificación del problema. Es un lenguaje para el procesamiento de información simbólica. PROLOG es una realización aproximada del modelo de computación de Programación Lógica sobre una máquina secuencial. Desde luego, no es la única realización posible, pero sí es una buena elección práctica, ya que equilibra por un lado la preservación de las propiedades del modelo abstracto de Programación Lógica y por el otro lado consigue que la implementación sea eficiente.

El lenguaje PROLOG juega un importante papel dentro de la Inteligencia Artificial, y se propuso como el lenguaje nativo de las máquinas de la quinta generación ("Fifth Generation Kernel Language", FGKL) que se quería que fueran Sistemas de Procesamiento de Conocimiento. La expansión y el uso de este lenguaje propició la aparición de la normalización del lenguaje Prolog con la norma ISO (propuesta de junio de 1993).

PROLOG es un lenguaje de programación para ordenadores que se basa en el lenguaje de la Lógica de Primer Orden y que se utiliza para resolver problemas en los que entran en juego *objetos* y *relaciones* entre ellos. Por ejemplo, cuando decimos "Jorge tiene una moto", estamos expresando una relación entre un objeto (Jorge) y otro objeto en particular (una moto). Más aún, estas relaciones tienen un orden específico (Jorge posee la moto y no al contrario). Por otra parte, cuando realizamos una pregunta (¿Tiene Jorge una moto?) lo que estamos haciendo es indagando acerca de una relación. Además, también solemos usar reglas para describir relaciones: "dos personas son hermanas si ambas son hembras y tienen los mismos padres". Como veremos más adelante, esto es lo que hacemos en Prolog.

Una de las ventajas de la programación lógica es que se especifica *qué* se tiene que hacer (*programación declarativa*), y no *cómo* se debe hacer (*programación imperativa*). A pesar de esto, Prolog incluye algunos predicados predefinidos meta-lógicos, ajenos al ámbito de la Lógica de Primer Orden, (var, nonvar, ==, ...), otros extra-lógicos, que tienen un efecto lateral, (write, get, ...) y un tercer grupo que nos sirven para expresar información de control de como realizar alguna tarea (el corte, ...). Por tanto, Prolog ofrece un sistema de programación práctico que tiene algunas de las ventajas de claridad y declarativiad que ofrecería un lenguaje de programación lógica y, al mismo tiempo, nos permite un cierto control y operatividad.

1. PROLOG Y EL LENGUAJE DE LA LÓGICA DE PRIMER ORDEN

La Lógica de Primer Orden analiza las frases sencillas del lenguaje (fórmulas atómicas o elementales) separándolas en *Términos* y *Predicados*. Los términos hacen referencia a los objetos que intervienen y los predicados a las propiedades o relaciones entre estos objetos. Además, dichas fórmulas atómicas se pueden combinar mediante *Conectivas* permitiéndonos construir frases más complejas (fórmulas moleculares).

1.1. PREDICADOS

Se utilizan para expresar propiedades de los objetos, *predicados monádicos*, y relaciones entre ellos, *predicados poliádicos*. En Prolog los llamaremos **hechos**. Debemos tener en cuenta que:

- Los nombres de todos los objetos y relaciones deben comenzar con una letra minúscula.
- Primero se escribe la relación o propiedad: *predicado*
- Y los objetos se escriben separándolos mediante comas y encerrados entre paréntesis: *argumentos*.
- Al final del hecho debe ir un punto ("**.**").

simbolo_de_predicado (arg1, arg2, ..., argn).

Tanto para los símbolos de predicado como para los argumentos, utilizaremos en Prolog constantes atómicas.

Ejemplos (`ej01.pl`):

```
/* Predicados monádicos: PROPIEDADES */

/* mujer(Per) <- Per es una mujer    */
mujer(clara).
mujer(chelo).

/* hombre(Per) <- Per es un hombre   */
hombre(jorge).
hombre(felix).
hombre(borja).

/* moreno(Per) <- Per tiene el pelo de color oscuro */
moreno(jorge).


/* Predicados poliádicos: RELACIONES */

/* tiene(Per,Obj) <- Per posee el objeto Obj */
tiene(jorge,moto).

/* le_gusta_a(X,Y) <- a X le gusta Y */
le_gusta_a(clara,jorge).
le_gusta_a(jorge,clara).
le_gusta_a(jorge,informatica).
le_gusta_a(clara,informatica).

/* es_padre_de(Padre,Hijo-a) <- Padre es el padre de Hijo-a */
es_padre_de(felix,borja).
es_padre_de(felix,clara).

/* es_madre_de(Madre,Hijo-a) <- Madre es la madre de Hijo-a */
es_madre_de(chelo,borja).
es_madre_de(chelo, clara).

/* regala(Per1,Obj,Per2) <- Per1 regala Obj a Per2 */
regala(jorge, flores, clara).
```

1.2. TÉRMINOS

Los términos pueden ser *constantes* o *variables*, y suponemos definido un dominio no vacío en el cual toman valores (Universo del Discurso). Para saber cuántos individuos del universo cumplen una determinada propiedad o relación *cuantificamos* los términos.

Las **constantes** se utilizan para dar nombre a objetos concretos del dominio, dicho de otra manera, representan individuos conocidos de nuestro Universo. Además, como ya hemos dicho, las constantes atómicas de Prolog también se utilizan para representar propiedades y relaciones entre los objetos del dominio. Hay dos clases de constantes:

- *Átomos*: existen tres clases de constantes atómicas:
  - Cadenas de letras, dígitos y subrayado (_) empezando por letra minúscula.
  - Cualquier cadena de caracteres encerrada entre comillas simples (').
  - Combinaciones especiales de signos: "?-", ":-", ...

- *Números*: se utilizan para representar números de forma que se puedan realizar operaciones aritméticas. Dependen del ordenador y la implementación[4].
  - *Enteros*: en la implementación de Prolog-2 puede utilizarse cualquier entero que el intervalo $[-2^{23},2^{23}-1]=[-8.388.608,8.388.607]$.
  - *Reales*: decimales en coma flotante, consistentes en al menos un dígito, opcionalmente un punto decimal y más dígitos, opcionalmente E, un + o - y más dígitos.

Ejemplos de constantes:

átomos válidos	átomos no válidos	números válidos	nº no válidos
f	2mesas	-123	123-
vacio	Vacio	1.23	.2
juan_perez	juan-perez	1.2E3	1.
'Juan Perez'	_juan	1.2E+3	1.2e3
a352	352a	1.2E-3	1.2+3

Las **variables** se utilizan para representar objetos cualesquiera del Universo u objetos desconocidos en ese momento, es decir, son las incógnitas del problema. Se diferencian de los átomos en que empiezan siempre con una letra mayúscula o con el signo de subrayado (_). Así, deberemos ir con cuidado ya que cualquier identificador que empiece por mayúscula, será tomado por Prolog como una variable. Para trabajar con objetos desconocidos cuya identidad no nos interesa, podemos utilizar la *variable anónima* (_). Las variables anónimas no están compartidas entre sí.

Ejemplos de variables:

```
X
Sumando
Primer_factor
_indice
_                    (variable anónima)
```

[4] Los ordenadores, vía hardware, resuelven eficientemente el manejo de los números y de las operaciones aritméticas, por tanto, en la práctica, la programación lógica lo deja en sus manos, trabajando en una aritmética estándar independiente del lenguaje.

Una variable está *instanciada* cuando existe un objeto determinado representado por ella. Y está *no instanciada* cuando todavía no se sabe lo que representa la variable. Prolog no soporta asignación destructiva de variables, es decir, cuando una variable es instanciada su contenido no puede cambiar. Como veremos más adelante, la manipulación de datos en la programación lógica se realiza por medio de la unificación.

Explícitamente Prolog no utiliza los símbolos de **cuantificación** para las variables, pero implícitamente sí que lo están, dependiendo de su ubicación. Las variables que aparecen en los hechos y las reglas están cuantificadas universalmente, y las variables que aparecen en las preguntas están cuantificadas existencialmente.

Ejemplo:

`le_gusta_a(jorge,X)`	significa que a Jorge le gustan todas las cosas
`?- le_gusta_a(jorge,X)`	pregunta si existe algo que le guste a Jorge

1.3. CONECTIVAS LÓGICAS

Puede que nos interese trabajar con sentencias más complejas, fórmulas moleculares, que constarán de fórmulas atómicas combinadas mediante conectivas. Las conectivas que hemos estudiado en la Lógica de Primer Orden son: *conjunción*, *disyunción*, *negación* e *implicación*.

La **conjunción**, "y", la representaremos poniendo una coma entre los objetivos "**,**" y consiste en *objetivos* separados que Prolog debe satisfacer, uno después de otro:

X , Y

Cuando se le da a Prolog una secuencia de objetivos separados por comas, intentará satisfacerlos por orden, buscando objetivos coincidentes en la Base de Conocimiento. Para que se satisfaga la secuencia se tendrán que satisfacer todos los objetivos.

La **disyunción**, "o", tendrá éxito si se cumple alguno de los objetivos que la componen. Se utiliza un punto y coma ";" colocado entre los objetivos:

X ; Y

La disyunción lógica también la podemos representar poniendo cada miembro de la disyunción en una cláusula aparte, como podemos ver en el ejemplo `es_hijo_de`.

La **negación**, "no", tendrá éxito si el objetivo *X* fracasa. No es una verdadera negación, en el sentido de la Lógica, sino una *negación "por fallo"*. La representamos con el predicado predefinido *not* o \+:

not(X) o \+ *X*

La **implicación** o **condicional**, sirve para significar que un hecho depende de un grupo de otros hechos. En castellano solemos utilizar las palabras “si ... entonces ...”. En Prolog se usa el símbolo “:-” para representar lo que llamamos una **regla**:

cabeza_de_la_regla :- cuerpo_de_la_regla.

La cabeza describe el hecho que se intenta definir; el cuerpo describe los objetivos que deben satisfacerse para que la cabeza sea cierta. Así, la regla:

C :- O_1, O_2, ..., O_n.

puede ser leída declarativamente como:

“La demostración de la cláusula C se sigue de la demostración de los objetivos O_1, O_2, ..., O_n.”

o procedimentalmente como:

“Para ejecutar el procedimiento C, se deben llamar para su ejecución los objetivos O_1,O_2,...,O_n.”

Otra forma de verla es como una implicación lógica “al revés”:

cuerpo_de_la_regla → cabeza_de_la_regla

Un mismo nombre de variable representa el mismo objeto siempre que aparece en la regla. Así, cuando *X* se instancia a algún objeto, todas las *X* de dicha regla también se instancian a ese objeto (*ámbito de la variable*).

Por último, daremos una serie de definiciones. Llamaremos **cláusulas** de un predicado tanto a los hechos como a las reglas. Una colección de cláusulas forma una BASE DE CONOCIMIENTO o BASE DE CONOCIMIENTOS.

Ejemplos (`ej01.pl`):

```
/* Conjunción de predicados   */
le_gusta_a(clara,jorge), le_gusta_a(clara,chocolate).

/* Disyunción de predicados   */
le_gusta_a(clara,jorge); le_gusta_a(jorge,clara).

/* Negación de predicados     */
not(le_gusta_a(clara,jorge)).
/+ le_gusta_a(clara,jorge).

/* Condicional: REGLAS        */

/* novios(Per1,Per2) <- Per1 y Per2 son novios */
novios(X,Y)    :-     le_gusta_a(X,Y),
                      le_gusta_a(Y,X).

/* hermana_de(Per1,Per2) <- Per1 es la hermana de Per2 */
hermana_de(X,Y)  :-  mujer(X),
                     es_padre_de(P,X), es_madre_de(M,X),
                     es_padre_de(P,Y), es_madre_de(M,Y).

/* Ejemplo de disyunción con ; y con diferentes cláusulas: */

/* 1. con ; (punto y coma)                                  */
es_hijo_de(X,Y)  :-  (es_padre_de(Y,X) ; es_madre_de(Y,X)).
```

```
/* 2. con cláusulas diferentes:                          */
es_hijo_de(X,Y)  :-  es_padre_de(Y,X).
es_hijo_de(X,Y)  :-  es_madre_de(Y,X).
```

2. ESTRUCTURA DE UN PROGRAMA

El hecho de programar en Prolog consiste en dar al ordenador un Universo finito en forma de hechos y reglas, proporcionando los medios para realizar inferencias de un hecho a otro. A continuación, si se hacen las preguntas adecuadas, Prolog buscará las respuestas en dicho Universo y las presentará en la pantalla. La programación en Prolog consiste en:

- declarar algunos HECHOS sobre los objetos y sus relaciones,
- definir algunas REGLAS sobre los objetos y sus relaciones, y
- hacer PREGUNTAS sobre los objetos y sus relaciones.

Programa Prolog: Conjunto de afirmaciones (*hechos* y *reglas*) representando los conocimientos que poseemos en un determinado dominio o campo de nuestra competencia.

Ejecución del programa: Demostración de un Teorema en este Universo, es decir, demostración de que una conclusión se deduce de las premisas (afirmaciones previas).

Programa Prolog
Base de Conocimientos + Motor de Inferencia

Un sistema Prolog está basado en un comprobador de teoremas por resolución para *cláusulas de Horn*. La regla de resolución no nos dice que cláusulas elegir ni que literales unificar dentro de cada cláusula. La estrategia de resolución particular que utiliza Prolog es una forma de *resolución de entrada lineal* (árbol de búsqueda estándar). Para la búsqueda de cláusulas alternativas para satisfacer el mismo objetivo, Prolog adopta una estrategia de *primero hacia abajo* (recorrido del árbol en profundidad). Por todo esto, el orden de las cláusulas (hechos y reglas) de un determinado procedimiento es importante en Prolog, ya que determina el orden en que las soluciones serán encontradas, e incluso puede conducir a fallos en el programa. Más importante es, si cabe, el orden de las metas a alcanzar dentro del cuerpo de una regla.

2.1. PREGUNTAS

Las preguntas, metas u objetivos son las herramientas que tenemos para recuperar la información desde Prolog. Al hacer una pregunta a un programa lógico queremos determinar si esa pregunta es *consecuencia lógica* del programa. Prolog considera que todo lo que hay en la Base de Conocimiento es verdad, y lo que no, es falso. De manera que si Prolog responde “yes” es que ha podido demostrarlo, y si responde “no” es que no lo ha podido demostrar (no debe interpretarse como “falso” si no que con lo que Prolog conoce no puede demostrar su veracidad).

Cuando se hace una pregunta a Prolog, éste efectuará una búsqueda por toda la Base de Conocimiento intentando encontrar hechos que *coincidan* con la pregunta. Dos hechos "coinciden" (se pueden unificar) si sus predicados son el mismo (se escriben de igual forma) y si cada uno de los respectivos argumentos son iguales entre sí.

?- simbolo_de_predicado (arg_1, arg_2, ..., arg_n).

Ejemplos (`ej01.pl`):

```
?- le_gusta_a(clara,jorge).
yes

?- le_gusta_a(jorge,cafe).
no

?- capital_de(madrid,españa).
no
```

Cuando a Prolog se le hace una pregunta con una variable, dicha variable estará inicialmente no instanciada. Prolog entonces recorre la Base de Conocimiento en busca de un hecho que *empareje* con la pregunta: los símbolos de predicado y el número de argumentos sean iguales, y emparejen los argumentos. Entonces Prolog hará que la variable se instancie con el argumento que esté en su misma posición en el hecho. Prolog realiza la búsqueda por la Base de Conocimiento en el orden en que se introdujo. Cuando encuentra un hecho que empareje, saca por pantalla los objetos que representan ahora las variables, y marca el lugar donde lo encontró. Prolog queda ahora a la espera de nuevas instrucciones, sacando el mensaje "More (y/n) ?":

- si pulsamos "n"+ <↵> cesará la búsqueda,
- si pulsamos "y"+<↵> reanudará la búsqueda comenzando donde había dejado la marca; decimos que Prolog está intentando *resatisfacer* la pregunta.

Otros compiladores simplemente se quedan a la espera y si pulsamos ";" nos muestran otra respuesta y si pulsamos <↵> finalizará.

Ejemplos (`ej01.pl`):

```
?- le_gusta_a(jorge,X).
X=clara
More (y/n)? y
X=informatica
More (y/n)? y
no
```

La conjunción y el uso de variables pueden combinarse para hacer preguntas muy interesantes:

```
?- le_gusta_a(clara,jorge),le_gusta_a(jorge,cafe).
no
?- le_gusta_a(clara,X),le_gusta_a(jorge,X).
X=informatica
More (y/n)? n
yes
```

Hemos buscado algo que le guste tanto a Clara como a Jorge. Para ello, primero averiguamos si hay algo que le guste a Clara, marcándolo en la Base de Conocimiento e instanciando la variable *X*. Luego, averiguamos si a Jorge le gusta ese objeto *X* ya instanciado. Si el segundo objetivo no se satisface, Prolog intentará resatisfacer el primer objetivo. Es importante recordar que cada objetivo guarda su propio marcador de posición.

```
?- le_gusta_a(clara,informatica); le_gusta_a(jorge,cafe).
yes

?- le_gusta_a(clara,cafe); le_gusta_a(jorge,cafe).
no

?- le_gusta_a(clara,X); le_gusta_a(jorge,X).
X=jorge
More (y/n)? y
X=informatica
More (y/n)? y
X=clara
More (y/n)? y
X=informatica
More (y/n)? y
no

?- not(le_gusta_a(clara,jorge)).
no

?- not(le_gusta_a(jorge,cafe)).
yes

?- hermana_de(clara,borja).
yes

?- hermana_de(borja,X).
no

?- hermana_de(clara,X).
X=borja
More (y/n)? n
yes
```

Ejercicio:

Piensa que ocurriría si a la pregunta de que si queremos más soluciones le contestamos que sí. ¿ Cómo lo solucionarías ?

3. SINTAXIS

La sintaxis de un lenguaje describe la forma en la que se nos está permitido juntar palabras entre sí. Los programas en Prolog se construyen a partir de **términos**. Un término es una *constante*, una *variable* o una *estructura*. Todo término se escribe como una secuencia de

caracteres. Para escribir un comentario lo encerraremos entre los signos /* y */ o desde el símbolo *%* hasta el final de línea. Así, Prolog pasa por alto los comentarios, pero los debemos añadir a nuestros programas para aclararlos y que el propio programa quede documentado (Ver apartado 7.2 ESTILO DE PROGRAMACIÓN EN PROLOG).

/ ... comentario ... */*

% Comentario de una sola línea

Ejemplo :

```
/* Esto es un comentario
   de más de una línea    */

% mujer(Per) <- Per es una mujer
mujer(clara).  % Esto es también comentario
mujer(chelo).
```

3.1. CARACTERES

Los nombres de constantes y variables se construyen a partir de cadenas de caracteres. Prolog reconoce dos tipos de caracteres:

* *Imprimibles*: hacen que aparezca un determinado signo en la pantalla del ordenador. Se dividen en cuatro categorías:
  letras mayúsculas: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.
  letras minúsculas: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.
  dígitos numéricos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
  signos: ! " # $ % & ' ( ) = - ^ | / \ { } [ ] _ @ + * ; : < > , . ?

* *No imprimibles*: no aparecen en forma de signo en la pantalla, pero realizan una determinada acción: nueva línea, retorno de carro, ...

Cada carácter tiene un entero entre 0 y 127 asociado a él, este es su código ASCII ("American Standard Code for Information Interchange").

3.2. ESTRUCTURAS

Una **estructura** es un único objeto que se compone de una colección de otros objetos, llamados componentes, lo que facilita su tratamiento. Una estructura se escribe en Prolog especificando su *nombre*, y sus *componentes*. Las componentes están encerradas entre paréntesis y separadas por comas; el nombre se escribe justo antes de abrir el paréntesis:

$$nombre\ (\ comp_1, comp_2, ..., comp_n\ )$$

Por ejemplo, podemos tener una estructura con el nombre libro, que tiene tres componentes: titulo, autor y año de edición. A su vez el autor puede ser una estructura con dos componentes: nombre y apellido:

```
libro(logica_informatica, autor(jose, cuena), 1985)
```

Como se puede ver, en Prolog, la sintaxis para las estructuras es la misma que para los hechos. Como podrás comprobar cuando trabajes más a fondo en Prolog, hay muchas ventajas en representar incluso los mismos programas Prolog como estructuras.

3.3. OPERADORES

En Prolog están predefinidos los **operadores aritméticos** y **relacionales** típicos, con la precedencia habitual entre ellos:

```
        ^
      mod
      *  /
      +  -
= \=  =<  >=  <  >
```

Para poder leer expresiones que contengan operadores necesitamos conocer los siguientes atributos:

* POSICIÓN: *Prefijo*: el operador va delante de sus argumentos.
Infijo: el operador se escribe entre los argumentos.
Postfijo: el operador se escribe detrás de sus argumentos.

* PRECEDENCIA: Nos indica el orden en que se realizan las operaciones. El operador más prioritario tendrá una precedencia 1 y el menos, 1201 (depende de la implementación).

*ASOCIATIVIDAD: Sirve para quitar la ambigüedad en las expresiones en las que hay dos operadores, uno a cada lado del argumento, que tienen la misma precedencia.

Para conocer las propiedades de los operadores ya definidos podemos utilizar el predicado predefinido:

current_op(Precedencia, Especificador, Nombre)

donde: *Precedencia* es un entero indicando la clase de precedencia,
Especificador es un átomo indicando la posición y la asociatividad, y
Nombre es un átomo indicando el nombre que queremos que tenga el operador.

Operador	Símbolo	Precedencia	Especificador
Potencia	^ ó **	200	xfx
Producto	*	400	yfx
División	/	400	yfx
División entera	//	400	yfx
Resto división entera	mod	400	yfx
Suma	+	500	yfx
Signo positivo	+	500	fx
Resta	-	500	yfx
Signo negativo	-	500	fx
Igualdad	=	700	xfx
Distinto	\=	700	xfx
Menor que	<	700	xfx
Menor o igual que	=<	700	xfx
Mayor que	>	700	xfx
Mayor o igual que	>=	700	xfx
Evaluación aritmética	is	700	xfx

Tabla 1: Operadores aritméticos y relacionales predefinidos en SWI-Prolog

Para la posición-asociatividad utilizamos átomos especiales del tipo:

xfx xfy yfy yfx xf yf fx fy

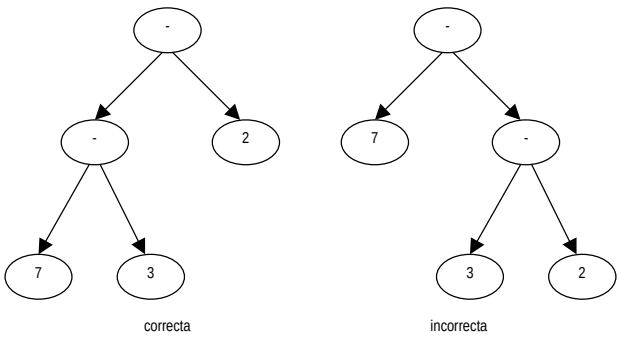
que nos ayudan a ver el uso del posible operador, representando **f** al operador, **x** un argumento indicando que cualquier operador del argumento debe tener una clase de precedencia estrictamente menor que este operador, e **y** un argumento indicando que puede contener operadores de la misma clase de precedencia o menor.

Ejemplo:

El operador - declarado como **yfx** determina que la expresión *a - b - c* sea interpretada como *(a - b) - c*, y no como *a - (b - c)* ya que la *x* tras la *f* exige que el argumento que va tras el primer - contenga un operador de precedencia estrictamente menor:

7 - 3 - 2 → (7 - 3) -2 = 4 - 2 = 2 correcta
→ 7 - (3 - 2) = 7 - 1 = 6 incorrecta

Gráficamente sería:



Si queremos declarar en Prolog un nuevo operador que sea reconocido por el sistema con una posición, clase de precedencia y asociatividad determinadas, utilizaremos el predicado predefinido *op*, cuya sintaxis es:

?- op(Precedencia,Especificador,Nombre).

Al trabajar con expresiones aritméticas y relacionales, los argumentos de estas estructuras deben ser constantes numéricas o variables instanciadas a constantes numéricas.

Ejemplo (`ej02.pl`):

```
/* horoscopo(Signo,DiaIni,MesIni,DiaFin,MesFin) <-
              pertenecen al signo del horoscopo Signo los nacidos
              entre el DiaIni del MesIni y el DiaFin del MesFin   */
horoscopo(aries,21,3,21,4).
horoscopo(tauro,21,4,21,5).
horoscopo(geminis,21,5,21,6).
horoscopo(cancer,21,6,21,7).
horoscopo(leo,21,7,21,8).
horoscopo(virgo,21,8,21,9).
horoscopo(libra,21,9,21,10).
horoscopo(escorpio,21,10,21,11).
horoscopo(sagitario,21,11,21,12).
horoscopo(capricornio,21,12,21,1).
horoscopo(acuario,21,1,21,2).
horoscopo(piscis,21,2,21,3).

/* signo(Dia,Mes,Signo) <- los nacidos el Dia de Mes pertenecen al
                           signo del zodiaco Signo                */
signo(Dia,Mes,Signo) :- horoscopo(Signo,D1,M1,D2,M2),
                        ( ( Mes=M1, Dia>=D1) ; ( Mes=M2, Dia=<D2) ).

?- signo(8, 5, tauro).
yes

?- signo(7, 8, acuario).
no

?- signo(7, 8, Signo).
Signo=leo
More (y/n)? y
no
```

Ejercicio:

Piensa qué ocurrirá si preguntamos `?- signo(7,X,Signo)`.

Y si preguntamos `?- signo(X,7,Signo)` . ¿ Por qué ?

El ejemplo contesta afirmativamente a preguntas del tipo `?- signo(74,4,tauro)`. Modifica el ejemplo para que trabaje con el número de días correcto para cada mes.

Los operadores no hacen que se efectúe ningún tipo de operación aritmética. El predicado de evaluación es el operador infijo *is*:

X is Y

donde: *Y* es un término que se interpreta como una expresión aritmética, con todos sus valores instanciados.
X puede ser una variable o una constante numérica.

Ejemplo (`ej03.pl`):

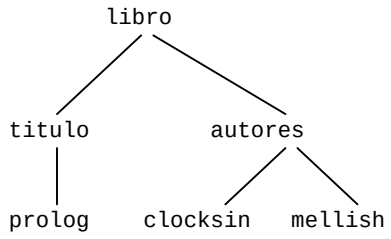
```
/* poblacion(Prov,Pob) <- la poblacion, en miles de habitantes,
                          de la provincia Prov es Pob            */
poblacion(alicante,1149).
poblacion(castellon,432).
poblacion(valencia,2066).

/* superficie(Prov,Sup) <- la superficie, en miles de km2, de la
                           provincia Prov es Sup                 */
superficie(alicante,6).
superficie(castellon,7).
superficie(valencia,11).

/* densidad(Prov,Den) <- la densidad de población, habitantes/km2,
                         de la provincia Prov es Den             */
densidad(X,Y) :- poblacion(X,P),
                 superficie(X,S),
                 Y is P/S.

?- densidad(alicante, X).
X=191'500

?- 6=4+2.
no
?- 6 is 4+2.
yes
?- N=4+2.
N=4+2
?- N is 4+2.
N=6
?- N is N+1.
no
?- N=4, N is 4+1.
no
```

4. ESTRUCTURAS DE DATOS

4.1. ÁRBOLES

Es más fácil entender la forma de una estructura complicada si la escribimos como un **árbol** en el que el nombre es un nodo y los componentes son las ramas.

Ejemplo: libro(titulo(prolog), autores(clocksin, mellish))



4.2. LISTAS

Las listas son unas estructuras de datos muy comunes en la programación no numérica. Una **lista** es una secuencia ordenada de elementos que puede tener cualquier longitud. Los elementos de una lista pueden ser cualquier término (constantes, variables, estructuras) u otras listas.

Las listas pueden representarse como un tipo especial de árbol. Una lista puede definirse recursivamente como:

* una **lista vacía []**, sin elementos, o
* una estructura con dos componentes:
  cabeza: primer argumento
  cola: segundo argumento, es decir, el resto de la lista.

El final de una lista se suele representar como una cola que contiene la lista vacía. La cabeza y la cola de una lista son componentes de una estructura cuyo nombre es "**.**".

Ejemplo:

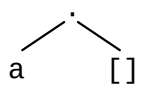
La lista que contiene un solo elemento *a* es

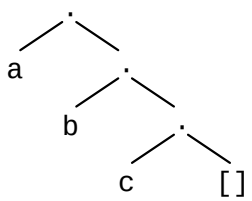
. (a, [])

y la lista de tres elementos *[a, b, c]* podría escribirse

. (a, . (b, . (c, [])))

siempre terminando con la lista vacía.





En Prolog utilizaremos una notación más sencilla de las listas que dispone a los elementos de la misma separados por comas, y toda la lista encerrada entre corchetes. Así, las

listas anteriores quedarían como *[a]* y *[a, b, c]*, que es un tipo de notación más manejable. Existe, tambien, una notación especial en Prolog para representar la lista con cabeza *X* (elemento) y cola *Y* (lista):

[X | Y]

Ejemplos:

Lista	Cabeza (elemento)	Cola (lista)
[a, b, c]	a	[b, c]
[a]	a	[ ]
[ ]	(no tiene)	(no tiene)
[[el, gato], maullo]	[el, gato]	[maullo]
[el, [gato, maullo]]	el	[[gato, maullo]]
[el, [gato, maullo], ayer]	el	[[gato, maullo], ayer]
[X+Y, x+y]	X+Y	[x+y]

Diseño de procedimientos de manejo de listas:

Vamos a utilizar la técnica de *refinamientos sucesivos* para el diseño de procedimientos de manejo de listas. Como no sabemos de antemano el tamaño de las listas deberemos utilizar la recursividad para recorrer las listas.

Ejemplo:

Procedimiento *miembro* que comprueba si un determinado elemento pertenece a una lista.

Esquema de la relación:

miembro(Elem, Lista) <- el término Elem pertenece a la lista Lista

Definición intuitiva:

Una determinada carta está en un mazo de cartas si es la primera o si está en el resto del mazo.

1ª aproximación: traducción literal

```
miembro(E,L) :- L=[X|Y]), X=E.
miembro(E,L) :- L=[X|Y]), miembro(E,Y).
```

2ª aproximación: recogida de parámetros

```
miembro(E,[X|Y]) :- X=E.
miembro(E,[X|Y]) :- miembro(E,Y).
```

3ª aproximación: unificación de variables

```
miembro(X,[X|Y]).
miembro(E,[X|Y]) :- miembro(E,Y).
```

4ª aproximación: ahorro de espacio en memoria (variable anónima)

```
miembro(X,[X|_]).
miembro(X,[_|Y]) :- miembro(X,Y).
```

Operaciones con listas: (ej04.pl)

```
/* miembro(Elem,Lista) <- el término Elem pertenece a la lista Lista */
miembro(X,[X|_]).
miembro(X,[_|Y]) :- miembro(X,Y).

/* nel(Lista,N) <- el numero de elementos de la lista Lista es N */
nel([],0).
nel([_|Y],N) :- nel(Y,M),
                N is M+1.

/* es_lista(Lista) <- Lista es una lista */
es_lista([]).
es_lista([_|_]).

/* concatena(L1,L2,L3) <- concatenación de las listas L1 y L2
                          dando lugar a la lista L3 */
concatena([],L,L).
concatena([X|L1],L2,[X|L3]) :- concatena(L1,L2,L3).

/* ultimo(Elem,Lista) <- Elem es el ultimo elemento de la lista Lista */
ultimo(X,[X]).
ultimo(X,[_|Y]) :- ultimo(X,Y).

/* inversa(Lista,Inver) <- Inver es la inversa de la lista Lista */
inversa([],[]).
inversa([X|Y],L) :- inversa(Y,Z),
                    concatena(Z,[X],L).

/* borrar(Elem,L1,L2) <- se borra el elemento Elem de la lista L1
                         obteniendose la lista L2 */
borrar(X,[X|Y],Y).
borrar(X,[Z|L],[Z|M]) :- borrar(X,L,M).

/* subconjunto(L1,L2) <- la lista L1 es un subconjunto de la lista L2 */
subconjunto([],_).
subconjunto([X|Y],Z) :- miembro(X,Z),
                        subconjunto(Y,Z).

/* insertar(Elem,L1,L2) <- se inserta el elemento Elem en la lista L1
                           obteniendose la lista L2 */
insertar(E,L,[E|L]).
insertar(E,[X|Y],[X|Z]) :- insertar(E,Y,Z).

/* permutacion(L1,L2) <- la lista L2 es una permutacion de la lista L1 */
permutacion([],[]).
permutación([X|Y],Z) :-      permutacion(Y,L),
                             insertar(X,L,Z).
```

Preguntas:

```
?- miembro(d,[a,b,c,d,e]).
yes

?- miembro(d,[a,b,c,[d,e]]).
no

?- miembro(d,[a,b,c]).
no

?- miembro(E,[a,b]).
E=a
More (y/n)? y
E=b
More (y/n)? y
no

?- nel([a,b,[c,d],e],N).
N=4

?- es_lista([a,b,[c,d],e]).
yes

?- concatena([a,b,c],[d,e],L).
L=[a,b,c,d,e]

?- concatena([a,b,c],L,[a,b,c,d,e]).
L=[d,e]

?- concatena(L1,L2,[a,b]).
L1=[], L2=[a,b]
More (y/n)? y
L1=[a], L2=[b]
More (y/n)? y
L1=[a,b], L2=[]
More (y/n)? y
no
```

Ejercicios:

1.- Comprueba el funcionamiento de las restantes operaciones con listas y prueba con diferentes tipos de ejemplos.

2.- Escribe un procedimiento que obtenga el elemento que ocupa la posición *n* de una lista o la posición que ocupa el elemento *e*:

```
?- elemento(E,3,[a,b,c,d]).
E=c

?- elemento(c,N,[a,b,c,d]).
N=3
```

3.- Modifica el procedimiento *borrar* para que borre el elemento que ocupa la posición *n* de la lista:

```
?- borrar(3,[a,b,c,d],L).
L=[a,b,d]
```

5. ESTRUCTURAS DE CONTROL

5.1. RECURSIÓN

Las definiciones recursivas se encuentran frecuentemente en los programas Prolog. Fijémonos en cómo algunos predicados de manejo de listas vistos en el punto anterior están definidos **recursivamente**, es decir, el cuerpo de la cláusula se llama a sí mismo. En la recursividad debemos tener cuidado en que se cumplan las "condiciones de límite" (punto de parada cuando utilizamos recursividad). Podemos observar que en la llamada de la cláusula recursiva al menos uno de los argumentos crece o decrece, para así poder unificar en un momento dado con la cláusula de la condición de parada.

Ejemplo:

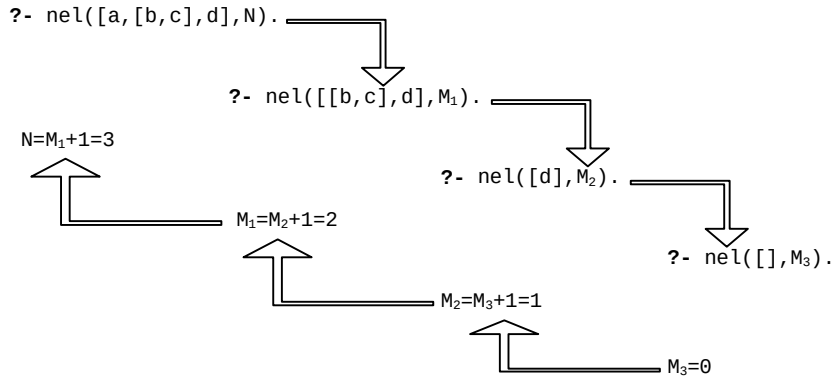
Procedimiento que calcula el número de elementos de una lista:

- *Condición de parada*: lista vacía
```
nel([],0).
```
- *Evolución de los parámetros*: lista con un elemento menos
```
nel([_|Y],N) :- nel(Y,M), N is M+1.
```

En la recursión encontramos dos partes:

- la primera parte en la que descendemos y construimos el árbol hasta encontrar el valor que unifica con la condición de parada
- una segunda parte en la que ascendemos por el árbol asignando valores a las variables que teníamos pendientes en las sucesivas llamadas.

Ejemplo:



N=3

```
?- trace,nel([a,[b,c],d],N),notrace.
   Call:  (  6) nel([a, [b, c], d], _G309) ? creep
   Call:  (  7) nel([[b, c], d], _L104) ? creep
   Call:  (  8) nel([d], _L117) ? creep
   Call:  (  9) nel([], _L130) ? creep
   Exit:  (  9) nel([], 0) ? creep
 ^ Call:  (  9) _L117 is 0 + 1 ? creep
 ^ Exit:  (  9) 1 is 0 + 1 ? creep
   Exit:  (  8) nel([d], 1) ? creep
 ^ Call:  (  8) _L104 is 1 + 1 ? creep
 ^ Exit:  (  8) 2 is 1 + 1 ? creep
   Exit:  (  7) nel([[b, c], d], 2) ? creep
 ^ Call:  (  7) _G309 is 2 + 1 ? creep
 ^ Exit:  (  7) 3 is 2 + 1 ? creep
   Exit:  (  6) nel([a, [b, c], d], 3) ? creep

N = 3
Yes

?- trace(nel),nel([a,[b,c],d],N).
        nel/2: call redo exit fail
T Call:  (  7) nel([a, [b, c], d], _G292)
T Call:  (  8) nel([[b, c], d], _L241)
T Call:  (  9) nel([d], _L254)
T Call:  ( 10) nel([], _L267)
T Exit:  ( 10) nel([], 0)
T Exit:  (  9) nel([d], 1)
T Exit:  (  8) nel([[b, c], d], 2)
T Exit:  (  7) nel([a, [b, c], d], 3)

N = 3
Yes
```

Deberemos tener cuidado de no escribir *recursiones circulares* (entrada en un bucle que no terminaría nunca) y de evitar la *recursión a izquierdas* (cuando una regla llama a un objetivo que es esencialmente equivalente al objetivo original), que causarían que Prolog no terminara nunca.

Recursión circular:

```
padre(X,Y) :- hijo(Y,X).
hijo(A,B) :- padre(B,A).
```

Recursión a izquierdas:

```
persona(X) :- persona(Y), madre(X,Y).
persona(adán).
```

5.2. UNIFICACIÓN

La **unificación** (“matching”) aplicada junto con la *regla de resolución* es lo que nos permite obtener respuestas a las preguntas formuladas a un programa lógico. La unificación constituye uno de los mecanismos esenciales de Prolog, y consiste en buscar instancias comunes a

dos átomos, uno de los cuales está en la cabeza de una cláusula y el otro en el cuerpo de otra cláusula. Prolog intentará hacerlos coincidir (*unificarlos*) mediante las siguientes reglas:

- Una variable puede instanciarse con cualquier tipo de término, y naturalmente con otra variable. Si *X* es una variable no instanciada, y si *Y* está instanciada a cualquier término, entonces *X* e *Y* son iguales, y *X* quedará *instanciada* a lo que valga *Y*. Si ambas están no instanciadas, el objetivo se satisface y las dos variables quedan *compartidas* (cuando una de ellas quede instanciada también lo hará la otra).

- Los números y los átomos sólo serán iguales a sí mismos. Evidentemente, también se pueden instanciar con una variable.

- Dos estructuras son iguales si tienen el mismo nombre y el mismo número de argumentos, y todos y cada uno de estos argumentos son unificables.

El predicado de igualdad (=) es un operador infijo que intentará unificar ambas expresiones. Un objetivo con el predicado no igual (\=) se satisface si el = fracasa, y fracasa si el = se satisface.

Ejemplos:

```
?- X=juan, X=Y.
X=juan, Y=juan

?- X=Y, X=juan.
X=juan, Y=juan

?- juan=juan.
yes

?- juan=pepe.
no

?- 1024=1024.
yes

?- amigo(pepe,juan)=amigo(pepe,X).
X=juan
```

```
?- amigo(pepe,juan)=Y.
Y=amigo(pepe,juan)

?- 9 \= 8.
yes

?- letra(C)=palabra(C).
no

?- 'juan'=juan.
yes

?- “juan”=juan.
no

?- f(X,Y)=f(A).
no
```

5.3. REEVALUACIÓN

La **reevaluación** (“backtracking” o *vuelta atrás*) consiste en volver a mirar lo que se ha hecho e intentar resatisfacer los objetivos buscando una forma alternativa de hacerlo. Si se quieren obtener más de una respuesta a una pregunta dada, puede iniciarse la reevaluación pulsando la tecla «*y*» cuando Prolog acaba de dar una solución y pregunta «More (y/n) ?», con lo que se pone en marcha el proceso de *generación de soluciones múltiples*.

Vamos a ver dos conceptos ya utilizados y relacionados directamente con la reevaluación:

- *Satisfacer* un objetivo: cuando Prolog intenta satisfacer un objetivo, busca en la Base de Conocimiento desde su comienzo, y:

  - si encuentra un hecho o la cabeza de una regla que pueda unificar con el objetivo, marca el lugar en la base de Conocimiento e instancia todas las variables previamente no instanciadas que coincidan. Si es una regla lo encontrado, intentará satisfacer los subobjetivos del cuerpo de dicha regla.

  - si no encuentra ningún hecho o cabeza de regla que unifique, el objetivo ha *fallado*, e intentaremos resatisfacer el objetivo anterior.

- *Resatisfacer* un objetivo: Prolog intentará resatisfacer cada uno de los subobjetivos por orden inverso, para ello intentará encontrar una cláusula alternativa para el objetivo, en cuyo caso, se dejan sin instanciar todas las variables instanciadas al elegir la cláusula previa. La búsqueda la empezamos desde donde habíamos dejado el marcador de posición del objetivo.

Ejemplo (`ej05.pl`):

```
/* animal(Anim) <- Anim es un animal */
animal(mono).
animal(araña).
animal(mosca).
animal(cocodrilo).

/* gusta(X,Y) <- a X le gusta Y */
gusta(mono,banana).
gusta(araña,mosca).
gusta(araña,hormiga).
gusta(cocodrilo,X) :- animal(X).
gusta(mosca,espejo).

/* regalo(X,Y) <- Y es un buen regalo para X */
regalo(X,Y) :- animal(X), gusta(X,Y).

?- regalo(X,Y).
X=mono, Y=banana
More (y/n)? y

X=araña, Y=mosca
More (y/n)? y

X=araña, Y=hormiga
More (y/n)? y

X=mosca, Y=espejo
More (y/n)? y

X=cocodrilo, Y=mono
More (y/n)? y

X=cocodrilo, Y=araña
More (y/n)? y
```

```
X=cocodrilo, Y=mosca
More (y/n)? y

X=cocodrilo, Y=cocodrilo
More (y/n)? y
no
```

Ejercicio:

Alterar el orden de los hechos de la base de conocimientos anterior, añadir nuevos hechos y/o reglas y estudiar como se van generando las sucesivas respuestas. Intenta entender por qué Prolog genera estas respuestas y en este determinado orden. Te puede ayudar si activas la opción de traza mediante el predicado predefinido *trace*, lo que te permitirá ir viendo los pasos que sigue prolog para obtener las distintas respuestas.

5.4. EL CORTE

El **corte** permite decirle a Prolog cuales son las opciones previas que no hace falta que vuelva a considerar en un posible proceso de reevaluación. Es un mecanismo muy delicado, y que puede marcar la diferencia entre un programa que funcione y uno que no funcione. Su utilidad vienen dada por:

- Optimización del tiempo de ejecución: no malgastando tiempo intentando satisfacer objetivos que de antemano sabemos que nunca contribuirán a una solución.

- Optimización de memoria: al no tener que registrar puntos de reevaluación para un examen posterior.

El corte se representa por el objetivo "!" que se satisface inmediatamente y no puede resatisfacerse de nuevo. Es decir, se convierte en una valla que impide que la reevaluación la atraviese en su vuelta hacia atrás, convirtiendo en inaccesibles todos los marcadores de las metas que se encuentran a su izquierda.

Se trata de una impureza en lo referente al control. Esto es, el control en Prolog es fijo y viene dado de forma automática por la implementación del lenguaje: recursión, unificación y reevaluación automática. De esta manera Prolog nos proporciona la ventaja de centrarnos en los aspectos lógicos de la solución, sin necesidad de considerar aspectos del control. Además, el corte puede conducirnos a programas difíciles de leer y validar.

Los usos comunes del corte se pueden dividir en tres áreas según la intención con la que lo hayamos colocado:

1).- CONFIRMACIÓN DE LA ELECCIÓN DE UNA REGLA: se informa a Prolog que va por buen camino, por tanto, si ha pasado el corte ha encontrado la regla correcta para resolver el objetivo. Si las metas siguientes fracasan, deberá fracasar el objetivo. El uso del corte con esta finalidad es semejante a simular un "if-then-else" (si *b* entonces *c* sino *d*):

```
a :- b, !, c.
a :- d.
```

que expresado sin utilizar el corte quedaría:

```
a :- b, c.
a :- not(b), d.
```

en la cual puede que Prolog acabe intentando satisfacer *b* dos veces. Así, debemos sopesar las ventajas de un programa claro sin cortes frente a uno que sea más rápido.

Ejemplo (`ej06.pl`):

```
/* sumatorio(Num,Sum) <- Sum es el sumatorio desde 1 hasta Num */
sumatorio(1,1) :- !.
sumatorio(N,S) :- N1 is N-1,
                  sumatorio(N1,S1),
                  S is N+S1.
```

Ejercicio:

Estudia que ocurriría si no hubiésemos puesto el corte en la primera cláusula.

2).- ADVERTENCIA DE QUE SE VA POR MAL CAMINO: se informa a Prolog que haga fracasar el objetivo sin intentar encontrar soluciones alternativas. Se utiliza en combinación con el predicado de fallo *fail*, el cual siempre fracasa como objetivo, lo que hace que se desencadene el proceso de reevaluación.

3).- TERMINACIÓN DE LA GENERACIÓN DE SOLUCIONES MÚLTIPLES: se informa a Prolog que deseamos finalizar la generación de soluciones alternativas mediante reevaluación.

Ejemplo (`ej06.pl`):

```
/* natural(Num) <- Num es un numero perteneciente a los Naturales */
natural(0).
natural(X) :- natural(Y),
              X is Y+1.

/* diventera(Dividendo,Divisor,Cociente) <- Cociente es el resultado
                 de la division entera de Dividendo entre Divisor */
diventera(A,B,C) :- natural(C),
                    Y1 is C*B,
                    Y2 is (C+1)*B,
                    Y1 =< A, Y2 > A, !.
```

Este programa Prolog va generando números naturales hasta encontrar un *C* que cumpla la condición:

$$B*C \leq A < B*(C+1)$$

siendo *A* el dividendo y *B* el divisor de la división. La regla *natural* va generando infinitos números naturales, pero sólo uno de ellos es el que nos interesa y será el que pase la condición. Así, el corte al final de la regla impide que en un posible proceso de reevaluación entremos en un bucle infinito: hemos encontrado la única solución y no hay ninguna razón para volver a buscar otra.

Discusión del "corte"

Mientras que un corte puede ser inofensivo, o incluso beneficioso, cuando se utiliza una regla de una forma dada, el mismo corte puede provocar un comportamiento extraño si la regla se utiliza de otra forma. Así, el siguiente predicado (`ej06.pl`):

```
/* concatena(L1,L2,L3) <- concatenación de las listas L1 y L2
                          dando lugar a la lista L3          */
concatena([],L,L) :- !.
concatena([X|L1],L2,[X|L3]) :- concatena(L1,L2,L3).
```

cuando consideramos objetivos de la forma

```
?- concatena([a,b,c],[d,e],X).
X=[a,b,c,d,e]
?- concatena([a,b,c],X,[a,b,c,d,e]).
X=[d,e]
```

el corte resulta muy adecuado, pero si tenemos el objetivo

```
?- concatena(X,Y,[a,b,c]).
X=[], Y=[a,b,c]
More (y/n)? y
no
```

ante la petición de nuevas soluciones, la respuesta es no, aunque de hecho existan otras posibles soluciones a la pregunta.

La conclusión sería que si se introducen cortes para obtener un comportamiento correcto cuando los objetivos son de una forma, no hay garantía de que los resultados sean razonables si empiezan a aparecer objetivos de otra forma. Sólo es posible utilizar el corte de una forma fiable si se tiene una visión clara de cómo se van a utilizar las reglas.

Ejemplos de uso del corte (`ej06.pl`):

```
/* borrar(Elem,L1,L2) <- L2 es la lista resultado de borrar todas las
                         ocurrencias del elemento Elem de la lista L1 */
borrar(_,[],[]).
borrar(E,[E|L1],L2) :- !, borrar(E,L1,L2).
borrar(E,[A|L1],[A|L2]) :- borrar(E,L1,L2).

/* sust(E1,E2,L1,L2) <- L2 es la lista resultado de sustituir en la lista
                        L1 todas las ocurrencias del elemento E1 por E2 */
sust(_,_,[],[]).
sust(E1,E2,[E1|L1],[E2|L2]) :- !, sust(E1,E2,L1,L2).
sust(E1,E2,[Y|L1],[Y|L2]) :- sust(E1,E2,L1,L2).

/* union(L1,L2,L3) <- L3 es la lista-conjunto unión de las
                      listas-conjuntos L1 y L2             */
union([],L,L).
union([X|L1],L2,L3) :- miembro(X,L2), !,
                       union(L1,L2,L3).
```

```
union([X|L1],L2,[X|L3]) :- union(L1,L2,L3).

?- borrar(a,[a,b,c,d,a,b,a,d],L).
L=[b,c,d,b,d]
?- sust(a,b,[a,b,c,d,a,b],L).
L=[b,b,c,d,b,b]
?- union([a,b,c],[a,d,f,b],L).
L=[c,a,d,f,b]
```

6. PREDICADOS DE ENTRADA Y SALIDA

Vamos a introducir una nueva ampliación sobre Prolog que no tiene nada que ver con la lógica. Hasta ahora, el único modo de comunicar información *a* y *desde* Prolog es a través del proceso de unificación entre nuestras preguntas y las cláusulas del programa. Pero puede que estemos interesados en una mayor interacción entre programa y usuario, aceptando entradas del usuario y presentando salidas al mismo. Así, aparecen los predicados de Entrada/Salida, que para mantener la consistencia del sistema cumplen:

- se evaluan siempre a verdad
- nunca se pueden resatisfacer: la reevaluación continua hacia la izquierda
- tiene un *efecto lateral* (efecto no lógico durante su ejecución): entrada o salida de un carácter, término, ...

6.1. LECTURA Y ESCRITURA DE TÉRMINOS

Escritura de términos	
write(X)	Si *X* está instanciada a un término, lo saca por pantalla. Si *X* no está instanciada, se sacará por pantalla una variable numerada de forma única (por ejemplo _69).
nl	Nueva Línea: la salida posterior se presenta en la siguiente línea de la pantalla.
tab(X)	Desplaza el cursor a la derecha *X* espacios. *X* debe estar instanciada a un entero.
display(X)	Se comporta como *write*, excepto que pasa por alto cualquier declaración de operadores que se haya hecho.

Lectura de términos	
read(X)	Lee el siguiente término que se teclee desde el ordenador. Debe ir seguido de un punto "." y un "retorno de carro". Instancia *X* al término leído, si *X* no estaba instanciada. Si *X* si que está instanciada, los comparará y se satisfará o fracasará dependiendo del éxito de la comparación.

Ejemplos (`ej07.pl`):

```
/* ibl(L) <- imprime la lista L de forma bonita, es decir
             saca por pantalla los elementos de la lista
             separados por un espacio y terminado en salto
             de línea                                      */
ibl([]) :- nl.
ibl([X|Y]) :-  write(X),
               tab(1),
               ibl(Y).

/* Diferencia entre write, display e ibl */
?- write(8+5*6), nl, display(8+5*6).
8+5*6
+(8,*(5,6))
yes

?- write(8*5+6), nl, display(8*5+6).
8*5+6
+(*(8,5),6)
yes

?- X=[esto,es,una,lista],write(X),nl,display(X),nl,ibl(X).
[esto,es,una,lista]
.(esto,.(es,.(una,.(lista,[]))))
esto es una lista
yes
```

6.2. LECTURA Y ESCRITURA DE CARACTERES

El carácter es la unidad más pequeña que se puede leer y escribir. Prolog trata a los caracteres en forma de enteros correspondientes a su código ASCII.

Escritura de caracteres	
put(X)	Si *X* está instanciada a un entero entre 0..255, saca por pantalla el carácter correspondiente a ese código ASCII. Si *X* no está instanciada o no es entero, error.

Lectura de caracteres	
get(X)	Lee caracteres desde el teclado, instanciando *X* al primer carácter imprimible que se teclee. Si *X* ya está instanciada, los comparará satisfaciéndose o fracasando.
get0(X)	Igual que el anterior, sin importarle el tipo de carácter tecleado.

Ejemplos (`ej07.pl`):

```
/* escribe_cadena(L) <- escribe en pantalla la lista L de
                        códigos ASCII en forma de cadena de
                        caracteres                          */
escribe_cadena([]).
escribe_cadena([X|Y]) :- put(X),
                         escribe_cadena(Y).

?- escribe_cadena([80,114,111,108,111,103]).
Prolog
yes

?- put(80),put(114),put(111),put(108),put(111),put(103).
Prolog
yes
```

6.3. LECTURA Y ESCRITURA EN FICHEROS

Existe un fichero predefinido llamado *user*. Al leer de este fichero se hace que la información de entrada venga desde el teclado, y al escribir, se hace que los caracteres aparezcan en la pantalla. Este el modo normal de funcionamiento. Pero pueden escribirse términos y caracteres sobre ficheros utilizando los mismos predicados que se acaban de ver. La única diferencia es que cuando queramos escribir o leer en un fichero debemos cambiar el *canal de salida activo* o el *canal de entrada activo*, respectivamente.

Posiblemente queramos leer y escribir sobre ficheros almacenados en discos magnéticos. Cada uno de estos tendrá un *nombre de fichero* que utilizamos para identificarlo. En Prolog los nombres de ficheros se representan como átomos. Los ficheros tienen una longitud determinada, es decir, contienen un cierto número de caracteres. Al final del fichero, hay una marca especial de *fin de fichero*. Cuando la entrada viene del teclado, puede generarse un fin de fichero tecleando el carácter de control ASCII 26 o control-Z. Se pueden tener abiertos varios ficheros, si conmutamos el canal de salida activo sin cerrarlos (`told`), permitiéndonos escribir sobre ellos en varios momentos diferentes, sin destruir lo escrito previamente.

Escritura sobre ficheros	
`tell(X)`	Si *X* está instanciada al nombre de un fichero, cambia el canal de salida activo. Crea un nuevo fichero con ese nombre. Si *X* no está instanciada o no es un nombre de fichero, producirá un error.
`telling(X)`	Si *X* no está instanciada, la instanciará al nombre del fichero que es el canal de salida activo. Si *X* está instanciada, se satisface si es el nombre del fichero actual de salida.
`told`	Cierra el fichero para escritura, y dirige la salida hacia la pantalla.

Lectura de ficheros	
see(X)	Si *X* está instanciada al nombre de un fichero, cambia el canal de entrada activo al fichero especificado. La primera vez que se satisface, el fichero pasa a estar abierto y empezamos al comienzo del fichero. Si *X* no está instanciada o no es un nombre de fichero, producirá un error.
seeing(X)	Si *X* no está instanciada, la instanciará al nombre del fichero que es el canal de entrada activo. Si *X* está instanciada, se satisface si es el nombre del fichero actual de entrada.
seen	Cierra el fichero para lectura, y dirige la entrada hacia el teclado.

Ejemplos:

```
/* Secuencia normal de escritura */
... , tell(fichero), write(X), told, ...

/* Conmutación de ficheros */
... , tell(fichero), write(A), tell(user), write(B),
      tell(fichero), write(C), told.

/* Secuencia normal de lectura */
... , see(fichero), read(X), seen, ...
```

7. PROGRAMACIÓN EN PROLOG

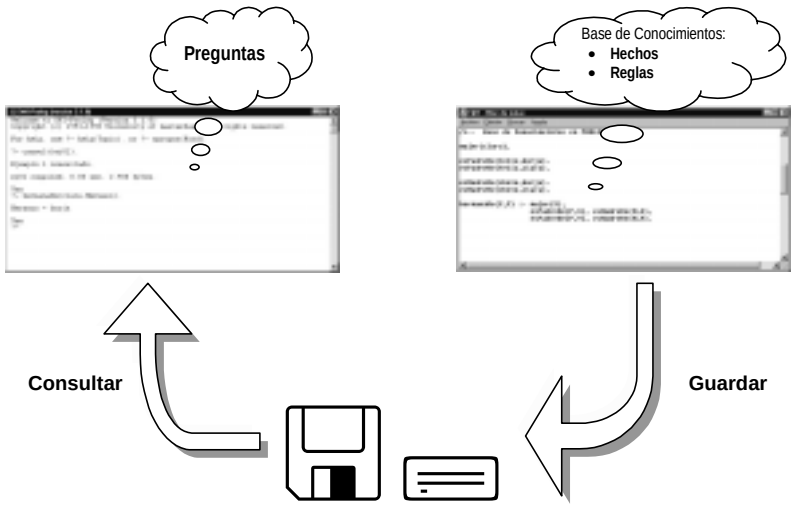
7.1. ENTORNO DE TRABAJO

En Prolog los ficheros se utilizan principalmente para almacenar programas y no perderlos al apagar el ordenador. Si tenemos el texto de un programa en un fichero, podemos leer todas las cláusulas del fichero y ponerlas en la Base de Conocimiento utilizando el predicado *consult*. Así, no tenemos que teclearlas cada vez que queramos trabajar con ese programa. Cuando *X* está instanciada al nombre de un fichero, leerá las cláusulas y reglas Prolog del fichero. Las nuevas cláusulas se añaden al final de las ya existentes en la Base de Conocimiento, sin destruir las anteriores. Al utilizar el predicado *reconsult*, si en el conjunto de las nuevas cláusulas hay algunas que se refieren a predicados que ya aparecían en la base de Conocimiento, estas serán reemplazadas por las nuevas. Resulta apropiado para corregir errores. La mayor parte de las implementaciones también permiten una notación especial que permite consultar una lista de ficheros, uno tras otro: la notación en forma de lista. Consiste en poner los nombres de los ficheros en una lista y darla como objetivo a satisfacer. Si se quiere consultar un fichero, se pone el nombre en la lista tal cual, mientras que si se quiere reconsultar se pone el nombre precedido de un signo "-".

Ejemplos:

```
?- consult(fichero).

fichero consulted

?- reconsult(fichero).

fichero reconsulted

?- [fichero1,-fichero2,-'prueba','a:prueba2'].

fichero1 consulted
fichero2 reconsulted
prueba reconsulted
a:prueba2 consulted
```

En la siguiente figura podemos ver un esquema de la forma de trabajo en Prolog. Mediante un editor de texto escribiremos nuestra *Base de Conocimientos* compuesta por distintos *hechos* y *reglas* que reflejan nuestro conocimiento acerca del problema a formalizar. Para poder utilizarlo deberemos guardarlo en disco (disco duro o disquete). Posteriormente desde Prolog deberemos consultar el fichero deseado. Una vez consultada la base de conocimientos correspondiente, estaremos en condiciones de hacer las *preguntas* de las que queramos obtener respuesta o que resuelvan el problema. Este ciclo se repetirá continuamente (editar-guardar-consultar-preguntar) hasta que demos por finalizado nuestro trabajo.



7.2. ESTILO DE PROGRAMACIÓN EN PROLOG

Los lenguajes de Programación Lógica, al igual que los lenguajes procedimentales, necesitan de una metodología para construir y mantener programas largos, así como un buen estilo de programación. Prolog ofrece mecanismos de control típicos de cualquier lenguaje imperativo y no puras propiedades o relaciones entre objetos. Dichos mecanismos contribuyen a

dar mayor potencia y expresividad al lenguaje, pero violan su naturaleza lógica. Adoptando un estilo apropiado de programación podemos beneficiarnos de esta doble naturaleza de Prolog.

- *Metodología de Diseño "top-down"*: descomponemos el problema en subproblemas y los resolvemos. Para ello solucionamos primero los pequeños problemas: *implementación "bottom-up"*. Esto nos va a permitir una mejor depuración ("debugged") de nuestros programas, ya que cada pequeño trozo de programa puede ser probado inmediatamente y, si lo necesita, corregido.

- Una vez analizado el problema y separado en trozos, el siguiente paso es decidir como representar y manipular tanto los objetos como las relaciones del problema. Debemos elegir los nombres de los predicados, variables, constantes y estructuras de manera que aporten claridad a nuestros programas (palabras o nombres mnemónicos que estén relacionados con lo que hacen).

- El siguiente paso es asegurarse de que la organización y la sintaxis del programa sea clara y fácilmente legible:

  - Llamamos *procedimiento* al conjunto de cláusulas para un predicado dado. Agruparemos por bloques los procedimientos de un mismo predicado (mismo nombre y mismo número de argumentos). Así, cada cláusula de un procedimiento comenzará en una línea nueva, y dejaremos una línea en blanco entre procedimientos.

  - Si el cuerpo de una regla es lo bastante corto, lo pondremos en una línea; sino, se escriben los objetivos de las conjunciones indentados y en líneas separadas.

  - Primero se escriben los hechos, y luego las reglas.

  - Es recomendable añadir comentarios y utilizar espacios y líneas en blanco que hagan el programa más legible. El listado del programa debe estar "autodocumentado". Debemos incluir para cada procedimiento y antes de las cláusulas el *esquema de la relación*.

  - Agrupar los términos adecuadamente. Dividir el programa en partes razonablemente autocontenidas (por ejemplo, todos los procedimientos de procesado de listas en un mismo fichero).

  - Evitar el uso excesivo del corte.

  - Cuidar el orden de las cláusulas de un procedimiento. Siempre que sea posible escribiremos las condiciones limite de la recursividad antes de las demás cláusulas. Las cláusulas "recogetodo" tienen que ponerse al final.

7.3. VENTAJAS DE PROLOG

Prolog, y en general los lenguajes de programación lógica, tienen las siguientes ventajas frente a los lenguajes clásicos (procedimentales):

- **Expresividad**: un programa (base de conocimiento) escrito en prolog puede ser leído e interpretado intuitivamente. Son, por tanto, más entendibles, manejables y fáciles de mantener.
- **Ejecución y búsqueda incorporada en el lenguaje**: dada una descripción prolog válida de un problema, automáticamente se obtiene cualquier conclusión válida.
- **Modularidad**: cada predicado (procedimiento) puede ser ejecutado, validado y examinado independiente e individualmente. Prolog no tiene variables globales, ni asignación. Cada relación está autocontenida, lo que permite una mayor *modularidad*, *portabilidad* y *reusabilidad* de relaciones entre programas.
- **Polimorfismo**: se trata de un lenguaje de programación sin tipos, lo que permite un alto nivel de abstracción e independencia de los datos (objetos).
- **Manejo dinámico y automático de memoria**.

8. PROGRAMA EJEMPLO: FORMAS NORMALES

El siguiente programa comprueba si una fórmula del Cálculo de Proposiciones es una *fórmula bien formada* (fbf) y la transforma a *Forma Normal Conjuntiva* (FNC) y a *Forma Normal Disyuntiva* (FND): (`ej08.pl`)

```
/*************************************************************/
/*                                                           */
/*  Departamento de Ciencia de la Computación e              */
/*                  Inteligencia Artificial                  */
/*  Universidad de Alicante                                  */
/*                                                           */
/*             LOGICA DE PRIMER ORDEN                        */
/*                    Prolog                                 */
/*                                                           */
/*  M. Jesús Castel                          Faraón Llorens  */
/*                                                           */
/*   S. O.      :  MS-DOS                                    */
/*   Interprete :  Prolog-2                                  */
/*   Fichero    :  EJ08.PL                                   */
/*   Programa   :  " FORMAS NORMALES "                       */
/*                     Lee  una fórmula del  Cálculo  de     */
/*                  Proposiciones,  comprueba si  es  una    */
/*                  Fórmula Bien Formada, y la transforma    */
/*                  a  Forma Normal Conjuntiva (FNC) y  a    */
/*                  Forma Normal Disyuntiva (FND).           */
/*                                                           */
/*                  Se lanza con :                           */
/*                    ?- menu.                               */
/*                                                           */
/*************************************************************/

 /*         Declaración de conectivas lógicas                */
?- op(10,fx,no).
?- op(100,yfx,o).
?- op(100,yfx,y).
?- op(200,yfx,imp).
```

```
?- op(200,yfx,coimp).

 /*-------------------------------------------------------------*/
 /*              PROGRAMA  PRINCIPAL                            */
menu :- cabecera,
        repeat,
          leer(Formula),
          hacer(Formula).

hacer(X) :- integer(X), X=0.
hacer('A') :- cabecera, ayuda, !, fail.
hacer('a') :- cabecera, ayuda, !, fail.
hacer(Formula) :- fbf(Formula),
                  formaNormal(Formula,FNC,FND),
                  escribirFormasNormales(FNC,FND),!, fail.
 /*                 Fin Programa Principal                     */
 /*-------------------------------------------------------------*/

 /*-------------------------------------------------------------*/
 /*        Determinar si es una Fórmula Bien Formada            */

/* fbf(Form) <- Form es una fórmula bien formada (fbf) del
                Cálculo de Proposiciones                       */
fbf((no P)) :- !,fbf(P).
fbf((P y Q)) :- !,fbf(P),fbf(Q).
fbf((P o Q)) :- !,fbf(P),fbf(Q).
fbf((P imp Q)) :- !,fbf(P),fbf(Q).
fbf((P coimp Q)) :- !,fbf(P),fbf(Q).
fbf(P) :- atom(P),!,proposicion(P).
fbf(_) :- nl,
          write('¡ ERROR ! : Formula erronea. Vuelve a introducirla.'),
          nl, nl, !, fail.

/* proposicion(Prop) <- Prop es una variable proposicional    */
proposicion(X) :- miembro(X,[p,q,r,s,t,u,v]),!.
proposicion(_) :- nl,
                  write('¡ERROR! no es simbolo de var. proposicional'),
                  nl,nl,!,fail.


 /*-------------------------------------------------------------*/
 /*              Transformar a Forma Normal                     */


/* formaNormal(Form,Fnc,Fnd) <- Fnc es la Forma Normal Conjuntiva
                       y Fnd es la Forma Normal Disyuntiva de Form */
formaNormal(Formula,FNC,FND) :- quitarImplicadores(Formula,F1),
                                normalizarNegadores(F1,F2),
                                exteriorizarConjuntor(F2,FNC),
                                exteriorizarDisyuntor(F2,FND).

/* quitarImplicadores(Form,Form1) <- Form1 es la fbf resultado de
             eliminar implicadores y coimplicadores a la fbf Form */
quitarImplicadores((P imp Q),((no P1) o Q1)) :-
                               !,quitarImplicadores(P,P1),
                               quitarImplicadores(Q,Q1).
quitarImplicadores((P coimp Q),(((no P1) o Q1) y ((no Q1) o P1))) :-
                               !,quitarImplicadores(P,P1),
                               quitarImplicadores(Q,Q1).
quitarImplicadores((P o Q),(P1 o Q1)) :-
                               !,quitarImplicadores(P,P1),
                               quitarImplicadores(Q,Q1).
```

```
quitarImplicadores((P y Q),(P1 y Q1)) :-
                              !,quitarImplicadores(P,P1),
                              quitarImplicadores(Q,Q1).
quitarImplicadores((no P),(no P1)) :- !,quitarImplicadores(P,P1).
quitarImplicadores(P,P).

/* normalizarNegadores(Form,Form1) <- Form1 es la fbf resultado de
                           normalizar los negadores a la fbf Form */
normalizarNegadores((no P),P1) :- !,negar(P,P1).
normalizarNegadores((P y Q),(P1 y Q1)) :- !,normalizarNegadores(P,P1),
                                          normalizarNegadores(Q,Q1).
normalizarNegadores((P o Q),(P1 o Q1)) :- !,normalizarNegadores(P,P1),
                                          normalizarNegadores(Q,Q1).
normalizarNegadores(P,P).

/* negar(Form,Form1) <- Form1 es la fbf resultado de negar la fbf
                        Form, teniendo en cuenta la Eliminación del
                        Doble Negador y  las leyes de De Morgan    */
negar((no P),P1) :- !,normalizarNegadores(P,P1).
negar((P y Q),(P1 o Q1)) :- !,negar(P,P1), negar(Q,Q1).
negar((P o Q),(P1 y Q1)) :- !,negar(P,P1), negar(Q,Q1).
negar(P,(no P)).

/* exteriorizarConjuntor(Form,Form1) <- Form1 es la fbf resultado
                    de exteriorizar los conjuntores de la fbf Form */
exteriorizarConjuntor((P o Q),R) :- !, exteriorizarConjuntor(P,P1),
                                    exteriorizarConjuntor(Q,Q1),
                                    fnc1((P1 o Q1),R).
exteriorizarConjuntor((P y Q),(P1 y Q1)) :-
                                    !, exteriorizarConjuntor(P,P1),
                                    exteriorizarConjuntor(Q,Q1).
exteriorizarConjuntor(P,P).

fnc1(((P y Q) o R),(P1 y Q1)) :- !, exteriorizarConjuntor((P o R),P1),
                                   exteriorizarConjuntor((Q o R),Q1).
fnc1((R o (P y Q)),(P1 y Q1)) :- !, exteriorizarConjuntor((R o P),P1),
                                   exteriorizarConjuntor((R o Q),Q1).
fnc1(P,P).

/* exteriorizarDisyuntor(Form,Form1) <- Form1 es la fbf resultado
                    de exteriorizar los Disyuntores de la fbf Form */
exteriorizarDisyuntor((P y Q),R) :- !, exteriorizarDisyuntor(P,P1),
                                    exteriorizarDisyuntor(Q,Q1),
                                    fnd1((P1 y Q1),R).
exteriorizarDisyuntor((P o Q),(P1 o Q1)) :-
                                    !, exteriorizarDisyuntor(P,P1),
                                    exteriorizarDisyuntor(Q,Q1).
exteriorizarDisyuntor(P,P).

fnd1(((P o Q) y R),(P1 o Q1)) :- !, exteriorizarDisyuntor((P y R),P1),
                                   exteriorizarDisyuntor((Q y R),Q1).
fnd1((R y (P o Q)),(P1 o Q1)) :- !, exteriorizarDisyuntor((R y P),P1),
                                   exteriorizarDisyuntor((R y Q),Q1).
fnd1(P,P).

 /*-------------------------------------------------------------*/
 /*              Pantalla de Ayuda                              */
ayuda :- nl,nl,write('    El programa lee una formula del Calculo de'),
         write('Proposiciones, comprueba que  es'),nl,
         write('una  Formula  Bien Formada (fbf), y la transforma a'),
         write(' Forma   Normal  Conjuntiva'),nl,
         write('(FNC) y a Forma Normal Disyuntiva (FND)'),nl,nl,
```

```
        write('    Las connectivas usadas son:'),nl,
        tab(20),write('no       negacion'),nl,
        tab(20),write('y        conjuncion'),nl,
        tab(20),write('o        disyuncion'),nl,
        tab(20),write('imp      implicador'),nl,
        tab(20),write('coimp    coimplicador'),nl,nl,
        nl.

 /*-------------------------------------------------------------*/
 /*              Procedimientos diversos                        */

cabecera :- nl,nl,write('Logica de Primer Orden              Dpto.'),
            write(' Tecnologia Informatica y Computacion'),nl,nl,
            write('Practicas PROLOG_____________________________'),
            write('_________________FORMAS NORMALES'),nl,nl.

/****   LEER lee una formula del Calculo de proposicions   ****/
leer(F) :- nl,write('  Introduce la formula terminada en punto (.)'),
           nl,write('        (  0  Terminar   /   A  Ayuda  )'),nl,
           nl,write('  FORMULA : '),
           read(F).

 /****   ESCRIBIR la formula en FNC y FND                  ****/
escribirFormasNormales(FNC,FND) :- write('FORMA NORMAL CONJUNTIVA:'),nl,
                                   write(FNC),nl,nl,
                                   write('FORMA NORMAL DISYUNTIVA:'),nl,
                                   write(FND).

/* miembro(Elem,Lista) <- el término Elem pertenece a la lista Lista */
miembro(X,[X|_]).
miembro(X,[_|Y]) :- miembro(X,Y).

 /*****************  FIN DEL PROGRAMA  ************************/
```

9. PREDICADOS PREDEFINIDOS

Vamos a ver algunos predicados predefinidos que ofrecen la mayoría de los sistemas Prolog.

1.- Adición de Bases de Conocimiento externas:

consult(X)	Añadir las cláusulas del fichero *X* a la base de Conocimiento
reconsult(Y)	Las cláusulas leídas de fichero *Y* sustituyen a las existentes para el mismo predicado
[X,-Y,Z]	Notación más cómoda para consult y reconsult
halt	Salir del sistema Prolog

2.- Construcción de objetivos compuestos (conectivas lógicas):

X,Y	Conjunción de objetivos
X;Y	Disyunción de objetivos
not(X)	Se cumple si fracasa el intento de satisfacer *X*

3.- Clasificación de términos:

var(X)	Se cumple si *X* es en ese momento una variable no instanciada
nonvar(X)	Se cumple si *X* no es una variable sin instanciar en ese momento
atomic(X)	Se cumple si *X* representa en ese momento un número o un átomo
atom(X)	Se cumple si *X* representa en ese momento un átomo de Prolog
numeric(X)	Se cumple si *X* representa en ese momento un número
integer(X)	Se cumple si *X* representa en ese momento un número entero
real(X)	Se cumple si *X* representa en ese momento un número real

4.- Control del programa:

true	Objetivo que siempre se cumple
fail	Objetivo que siempre fracasa
!	Corte. Fuerza al sistema Prolog a mantener ciertas elecciones
repeat	Sirve para generar soluciones múltiples mediante el mecanismo de reevaluación
call(X)	Se cumple si tiene éxito el intento de satisfacer *X* (instanciada a un término)

5.- Operadores aritméticos y relacionales:

X=Y	Se cumple si puede hacer *X* e *Y* iguales (unificarlos)
X\=Y	Se cumple si *X*=*Y* fracasa
X==Y	Igualdad más estricta
X\==Y	Se cumple si fracasa ==
X < Y	Predicado menor que
X > Y	Predicado mayor que
X >= Y	Predicado mayor o igual que
X =< Y	Predicado menor o igual que
X @< Y	Ordenación de términos. Predicado menor que
X @> Y	Ordenación de términos. Predicado mayor que
X @>= Y	Ordenación de términos. Predicado mayor o igual que
X @=< Y	Ordenación de términos. Predicado menor o igual que
X is Y	Se evalúa la expresión a la que esta instanciada *Y* para dar como resultado un número, que se intentará hacer coincidir con *X*
X is_string Y	Se evalúa la expresión a la que está instanciada *Y* para dar como resultado una cadena, que se intentará hacer coincidir con *X*
X & Y	Operador concatenación de cadenas de caracteres
X + Y	Operador suma
X - Y	Operador resta
X * Y	Operador de multiplicación
X / Y	Operador de división
X // Y	Operador de división entera
X mod Y	Operador de resto de la división entera
X ^ Y	Operador de exponenciación
op(X,Y,Z)	Declara el operador de nombre *Z* con clase de precedencia *X* y posición-asociatividad *Y*
current_op(X,Y,Z)	Busca el operador ya declarado de nombre *Z* con clase de precedencia *X* y posición-asociatividad *Y*

6.- Predicados de Entrada y Salida:

get0(X)	Se cumple si *X* puede hacerse corresponder con el siguiente carácter encontrado en el canal de entrada activo (en código ASCII)
get(X)	Se cumple si *X* puede hacerse corresponder con el siguiente carácter imprimible encontrado en el canal de entrada activo
skip(X)	Lee y pasa por alto todos los caracteres del canal de entrada activo hasta que encuentra un carácter que coincida con *X*
read(X)	Lee el siguiente término del canal de entrada activo
put(X)	Escribe el entero *X* como un carácter en el canal de salida activo
nl	Genera una nueva línea en el canal de salida activo
tab(X)	Escribe *X* espacios en el canal de salida activo
write(X)	Escribe el término *X* en el canal de salida activo
display(X)	Escribe el término *X* en el canal de salida activo, pasando por alto las declaraciones de operadores

7.- Manejo de ficheros:

see(X)	Abre el fichero *X* y lo define como canal de entrada activo
seeing(X)	Se cumple si el nombre del canal de entrada activo coincide con *X*
seen	Cierra el canal de entrada activo, y retorna al teclado
tell(X)	Abre el fichero *X* y lo define como canal de salida activo
telling(X)	Se cumple si *X* coincide con el nombre del canal de salida activo
told	Cierra con un fin de fichero el canal de salida activo, que pasa a ser la pantalla

8.- Manipulación de la Base de Conocimiento:

listing	Se escriben en el fichero de salida activo todas las cláusulas
listing(X)	Siendo *X* un átomo, se escriben en el fichero de salida activo todas las cláusulas que tienen como predicado dicho átomo
clause(X,Y)	Se hace coincidir *X* con la cabeza e *Y* con el cuerpo de una cláusula existente en la base de Conocimiento
assert(X)	Permite añadir la nueva cláusula *X* en la base de Conocimiento
asserta(X)	Permite añadir la nueva cláusula *X* al principio de la base de Conocimiento
assertz(X)	Permite añadir la nueva cláusula *X* al final de la base de Conocimiento
retract(X)	Permite eliminar la primera cláusula de la base de Conocimiento que empareje con *X*
abolish(X)	Retira de la Base de Conocimiento todas las cláusulas del predicado *X*

9.- Construcción y acceso a componentes de estructuras:

functor(E,F,N)	*E* es una estructura de nombre *F* y *N* argumentos
arg(N,E,A)	El argumento número *N* de la estructura *E* es *A*
E=..L	Predicado *univ*. *L* es la lista cuya cabeza es el nombre de la estructura *E* y la cola sus argumentos
name(A,L)	Los caracteres del átomo *A* tienen por ASCII los números de la lista *L*

10.- Depuración de programas Prolog:

trace	Activación de un seguimiento exhaustivo, generando la traza de la ejecución de las metas siguientes.
notrace	Desactiva el seguimiento exhaustivo
spy P	Fijación de puntos espía
nospy P	Elimina los puntos espía especificados
debugging	Ver qué puntos espía se han establecido hasta el momento
nodebug	Retira todos los puntos espía activos en ese momento

10. SISTEMAS PROLOG

Existen gran cantidad de implementaciones del lenguaje de programación lógica Prolog, tanto en versiones comerciales como de libre distribución. Por razones de facilitar el acceso a dichos sistemas, vamos a comentar dos sistemas Prolog de libre distribución para ordenadores PC, de forma que el lector puede acceder a ellos, instalarlos en su máquina y así ejecutar los ejemplos de este anexo (y lo más importante, probar todo aquello que se le ocurra).

Si se quiere una lista más extensa de implementaciones del lenguaje Prolog, con las direcciones donde localizarlas, se puede visitar la página Web:

http://gruffle.comlab.ox.ac.uk/archive/logic-prog.html

10.1. PROLOG-2

Prolog-2 es un sistema Prolog de Dominio Público de "Expert Systems Ltd". Es una implementación del Prolog de Edinburgh descrito por Clocksin y Mellish en el libro "Programación en Prolog", convertido ya en un estándar. Encontraremos mayor información sobre esta implementación en los libros "The Prolog-2 User Guide" y "The Prolog-2 Encyclopaedia" de Tony Dodd publicados por Ablex Publishing Corporation. El software lo podemos encontrar vía ftp en la siguiente dirección:

ftp://ai.uga.edu/pub/prolog

Su principal ventaja es que se puede trabajar sobre sistemas DOS muy sencillos. Se puede ejecutar tanto desde disco duro como desde disquete, tecleando:

```
prolog2
```

Si da problemas durante el arranque, comprobar que el fichero CONFIG.SYS contenga una línea que nos permita tener abiertos al menos 20 archivos. Si no, añadir la línea:

```
FILES=20
```

El sistema Prolog-2 posee un editor integrado, al que accedemos con el predicado predefinido *edit*. El fichero es automáticamente reconsultado después de su edición. Así:

```
?- edit(fichero).
```

edita el fichero *fichero.pl* y lo reconsulta. Sólo están libres para la edición 128.000 bytes. Si necesitamos más, la llamada al editor deberá ser del tipo:

```
?- edit(fichero,200000).
```

que dejara libres para edición 200.000 bytes de memoria. Se suministra el editor PrEd, el cual contiene ayuda en línea. Para más detalles de este editor consultar el archivo "pred.doc". Podemos utilizar cualquier otro editor que nosotros queramos, simplemente debemos modificar del archivo "edit.bat" la llamada a *pred* , ya que el predicado predefinido *edit* invoca al fichero batch "edit.bat" .

Para añadir cláusulas a Prolog podemos editar el fichero mediante *edit*, y al salir, como lo reconsulta automáticamente, quedan incorporadas a Prolog. También podemos añadir cláusulas mediante el predicado:

```
[user].
```

De esta manera entramos en el *modo de consulta*, indicado por el prompt "C:". Una vez tecleadas las cláusulas podemos volver al *modo de pregunta*, indicado por el prompt "?-", tecleando ^Z.

Las *teclas de función*, dentro del sistema Prolog-2, tienen el siguiente significado:

F1	Ayuda
F2	Menú
F3	Línea de comandos introducida anteriormente
F4	Línea siguiente
F5	Borrar
F6	Información sobre las teclas de función
F7	Ocultar
F8	Ocultar Debugger
F9	Ocultar todo
F10	Mostrar Debugger

10.2. SWI-Prolog

SWI-Prolog es un compilador Prolog de Dominio Público, que como el anterior sigue el estándar de Edinburgh. Ha sido diseñado e implementado por Jan Wielemaker, del departamento Social Science Informatics (SWI) de la Universidad de Amsterdam. El software lo podemos encontrar vía ftp en la siguiente dirección:

ftp://swi.psy.uva.nl/pub/SWI-Prolog

La plataforma original sobre la que se desarrollo fue para Unix, aunque en la actualidad podemos encontrar versiones para Linux y para PC que se ejecutan bajo el entorno Windows

(Windows 3.1., Windows 95 y Windows NT). Es potente y flexible (permite integración con C), y tiene las ventajas sobre el anterior de que, de momento, se mantiene actualizado y se ejecuta sobre un entorno gráfico más amigable y agradable para el usuario.

Posee un sistema de ayuda en línea que se puede consultar mediante los predicados:

```
?- help.
?- help(Termino).
?- apropos(Cadena).
```

También se puede conseguir una versión escrita del manual de referencia, en formato PostScript, en la dirección:

ftp://swi.psy.uva.nl/pub/SWI-Prolog/refman

Para ejecutarlo, una vez iniciado Windows, basta con hacer doble click sobre el icono correspondiente (plwin.exe).

Al estar desarrollado sobre plataforma Unix, reconoce algunas órdenes del sistema operativo que nos pueden ser de utilidad:

`cd`	cambiar el directorio de trabajo
`pwd`	muestra el directorio de trabajo
`ls`	muestra el contenido del directorio de trabajo

Bibliografía

Aquí presentamos una breve reseña bibliográfica. No pretende ser una bibliografía extensa sobre el tema y que abarque todas las publicaciones sobre este campo del saber. Simplemente contiene aquellos libros que los autores hemos consultado para la elaboración de este libro y cuya lectura consideramos de interés, y que el alumno tiene a su disposición en las bibliotecas de la Universidad de Alicante.

Lógica:

[1] Alchurrón, Carlos E.
Lógica
Editorial Trotta / Consejo Superior de Investigaciones Científicas, 1995

[2] Antón, Amador y Casañ, Pascual
Lógica Matemática. Ejercicios. I. Lógica de enunciados
Nau llibres, 1987

[3] Arenas Alegría, Lourdes
Lógica Formal para Informáticos
Ediciones Díaz de Santos, 1996

[4] Badesa, Calixto; Jané, Ignacio y Jansana, Ramón
Elementos de lógica formal
Ariel Filosofía, 1998

[5] Barwise, Jon y Etchemendy, John
The Language of First-Order Logic
including *Tarski's World 4.0*
CSLI lecture notes nº34, third edit.
CSLI Publ. Stanford, Calif., 1992

[6] Barwise, Jon y Etchemendy, John
Hyperproof
CSLI lecture notes nº 42, 1.994
CSLI Publ.., Stanford, California

[7] Boyer, Robert S. y Moore, J. S.
A Computational Logic Handbook
Academic Press, 1988

[8] Cohen, David E.
Computability and logic
Ellis Horwood Limited, 1987

[9] Cuena, José
Lógica Informática
Alianza Editorial, S.A., 1985

[10] Deaño, Alfredo
Introducción a la lógica formal
Alianza Universidad Textos, 1992

[11] Dijstra, E. W. y Scholten, C. S.
Predicate Calculus and Program Semantics
Springer-Verlag, 1990

[12] Eberle, Ruth
Instructor's Manual
The Language of First-Order Logic
CSLI Pub.., Stanford Univ. 1993

[13] Ershov, Yu. y Paliutin, E.
Lógica Matemática
Ed. Mir Moscú, 1990

[14] Fitting, Melvin
First Order Logic and Automated Theorem Proving
Springer-Verlag, 1990

[15] Garrido, Manuel
Lógica Simbólica
Editorial Tecnos, S.A., 2ºed. 1991

[16] Grassmann,W.K. y Tremblay, J.P.
Matemática Discreta y Lógica. Una perspectiva desde la Ciencia de la Computación
Editorial Prentice Hall, 1996

[17] Hamilton, A. G.
Lógica para Matemáticos
Paraninfo, S.A., 1981

[18] Kowalski, Robert A.
Logic for Problem Solving
The Computer Science Lib., 1979

[19] Kowalski, Robert A.
Lógica, programación e inteligencia artificial
Edic. Díaz de Santos, S.A., 1986

[20] Moore, Robert C.
Logic and Representation
CSLI Lecture Notes nº 39, 1995
CSLI Public., Stanford, California

[21] Mosterín, Jesús
Lógica de Primer Orden
Editorial Ariel, S.A., 1983

[22] Nerode, Anil y Shore, Richard A.
Logic for Applications
Springer-Verlag, 1997

[23] Peña, Lorenzo
Rudimentos de la Lógica Matemática
Consejo Sup. Inv. Científ. 1991

[24] Pérez Sedeño, Eulalia
Ejercicios de lógica
Siglo Veintiuno España Ed., 1991

[25] Reeves, S. y Clarke, M.
Logic for Computer Science
Addison-Wesley, 1990

[26] Richards, Tom
Clausal Form Logic. An introduction to the Logic of Computer Reasoning
Addison-Wesley Pub. Comp., 1989

[27] San Román, J. Sancho
Lógica Matemática y Computabilidad
Edic. Díaz de Santos, S.A., 1.990

[28] Smullyan, Raymond M.
First Order Logic
Springer-Verlag, 1968

[29] Sperschneider,V. y Antoniou, G.
Logic. A Foundation for Computer Science
Ed. Addison-Wesley, 1991

[30] Suppes, P. y Hill, S.
Introducción a la Lógica Matemática
Editorial Reverté, S.A., 1990

Programación Lógica y Prolog:

[1] Amble, Tore
Logic Programming and Knowledge Engineering
Ed. Addison-Wesley, 1987

[2] Berk, A.A.
PROLOG. Programación y aplicaciones en Intelig. Artificial.
Ed. ANAYA Multim., S.A., 1986

[3] Bratko, Ivan
PROLOG Programming for Artificial Intelligence
Addison-Wesley, 2ª ed., 1990

[4] Clocksin, William F.
Clause and Effect. Prolog Programming for the Working Programmer
Springer-Verlag, 1997

[5] Clocksin, W. F. y Mellish, C. S.
Programación en PROLOG
Ed. Gustavo Gili, S.A., 1987

[6] Covington, Michael A.
ISO Prolog. A Summary of the Draft Proposed Standard
A. I. Prog. Univ. of Georgia 1993

[7] Dodd, Tony
The Prolog-2 Encyclopaedia
Ablex Publishing Corporation

[8] Dodd, Tony
The Prolog-2 User Guide
Ablex Publishing Corporation

[9] Dodd, Tony
Prolog. A Logical Approach
Oxford University Press, 1990

[10] Giannesini, F., Kanoui, H., Pasero, R. y van Caneghem, M.
Prolog
Addison-Wesley Iberoamer., 1989

[11] Kim, Steven H.
Knowledge Systems Through PROLOG
Oxford University Press, 1991

[12] Kowalski,Bob y Kriwaczek, Frank
Logic Programming. Prolog and its applications
Addison-Wesley Pub. Co., 1986

[13] Lloyd, J. W.
Foundations of Logic Programming
Ed. Springer-Verlag, 1993

[14] Maier, David y Warren, David S.
Computing with Logic. Logic programming with Prolog
The Benjamin/Cummings, 1988

[15] O'Keefe, Richard A.
The Craft of Prolog
The MIT Press, Cambridge, 1990

[16] Sterling, Leon y Shapiro, Ehud
The Art of PROLOG. Advanced Programming Techniques
The MIT Press, 2ª ed. 1994

[17] Wielemaker, Jan
SWI-Prolog3.1.Reference Manual
Dept. of Social Science Informatics (SWI), Univ. of Amsterdam, 1998

[18] Special Issue: *Ten Years of Logic Programming*
Journal of Logic Programming, vol. 19/20, may/july 1994

Libros complementarios:

[1] Carroll, Lewis
Alicia en el Pais de las Maravillas
Alianza Editorial, 1994

[2] Carroll, Lewis
Alicia a través del Espejo
Alianza Editorial, 1993

[3] Carroll, Lewis
El juego de la lógica
Alianza Editorial, 1988

[4] Casti, John L.
El Quinteto de Cambridge. Una obra de especulación científica
Taurus Pensamiento, 1998

[5] Crevier, Daniel
Inteligencia Artificial
Acento Editorial, 1996

[6] Escher, M. C.
Estampas y dibujos
Benedikt Taschen, 1994

[7] Gaarder, Jostein
El mundo de Sofía. Novela sobre la história de la Filosofía
Editorial Siruela, 23ª ed., 1996

[8] Gardner, Martin
¡AJÁ! PARADOJAS. Paradojas que hacen pensar
Editorial Labor S.A., 1.989

[9] Gardner, Martin
Máquinas y Diagramas Lógicos
Libro bolsillo Alianza Edit. 1985

[10] González, Mª José
Introducción a la Psicología del Pensamiento
Editorial Trotta, 1998

[11] Hofstadter, Douglas R.
Gödel, Escher, Bach. Un Eterno y Grácil Bucle
Tusquets Editores, 6ª ed.,1998

[12] Luger, George F. (editor)
Computation & Intelligence. Collected Readings
The AAAI Press/The MIT Press, 1995

[13] Marx, Groucho
Groucho y Yo
Tusquets Editores, 3ª ed. 1998

[14] Nagel, E. y Newman, J.R.
El Teorema de Gödel
Ed. Tecnos, S.A. 2ª edic. 1994

[15] Paulos, John Allen
Pienso, luego río
Ediciones Cátedra S.A., 1994

[16] Penrose, Roger
La Nueva Mente del Emperador
Biblioteca Mondadori, 1991

[17] Smullyan, Raymond
¿Cómo se llama este libro?
Ediciones Cátedra S.A., 1989

[18] Smullyan, Raymond
Alicia en el País de las Adivinanzas
Ediciones Cátedra S.A., 1991

[19] Smullyan, Raymond
Satán, Cantor y el infiito
Gedisa Editorial, 1992