2 LÓGICA PROPOSICIONAL

En 1.2.4 se habló de la lógica como una parte de las matemáticas que se usa para razonar de manera formal. Los sistemas lógicos se han presentado como herramientas que deberán llevar a cabo la representación de afirmaciones y el razonar y deducir de manera simbólica.

La lógica proposicional es un primer tipo de sistema lógico, en el que se manejan afirmaciones con una estructura de complejidad bastante simple. Por ser así de simple, en la lógica proposicional es fácil definir y entender qué es ser verdad, cuándo una afirmación es cierta, cuándo vale en cualquier escenario, y muchos otros conceptos interesantes. Es un primer peldaño de una escala de sistemas lógicos en la que los peldaños más altos incluyen a los más bajos, ganando en expresividad pero también en complejidad.

2.1 LÓGICA PROPOSICIONAL

En 1.2.4 se mencionó que una *proposición* es una frase declarativa y que, como tal, es una afirmación sobre algo que puede o no ser verdad. No sobra decir que no puede ser verdadera y falsa al mismo tiempo.

Ejemplos de proposiciones:

- $1+1=2$
- $1+6 \leq 3$
- Juan es médico.
- Juan es médico y Pedro es arquitecto.

Las dos primeras son frases que se pueden entender como afirmaciones sobre números enteros. La tercera y la cuarta son afirmaciones sobre la profesión de unas personas. En todos los casos se puede decir si lo que se afirma es cierto o no.

Hay una diferencia estructural entre la tercera y la cuarta frases. De hecho, la tercera es parte de la cuarta, y esta última se puede ver como la afirmación de dos afirmaciones más simples. En otras palabras, hay *proposiciones simples*, que se van a considerar atómicas, sin estructura. Y *proposiciones complejas*, compuestas de proposiciones más simples unidas por alguna especie de elemento gramatical.

Por otro lado, no toda frase en lenguaje natural tiene que ser falsa o verdadera. Por eso, no toda frase puede ser una proposición. Ejemplos de frases que no son proposiciones:

- ¿Qué hora es?
- $x+1 = 3$
- Conduzca con cuidado.

La primera y la tercera no son frases declarativas. La segunda sí declara algo, pero su valor de verdad no se conoce mientras no se sepa el valor de x[1]. O sea, no son proposiciones y, como frases, ninguna es cierta o falsa.

En una proposición simple no se quieren ver partes que la compongan y es en este sentido que es atómica. Si se entiende que la frase "Juan es médico" es una proposición, una manera de representarla sería con algún identificador, que no es otra cosa que una palabra de uno o más símbolos en algún alfabeto, como podría ser `Juan_es_medico`. El identificador se está entendiendo como un elemento que denota la frase y,

[1] En desarrollos posteriores (cuando se hable de lógica de predicados) esta afirmación se podrá calificar de falsa o verdadera. Por ahora, no se quiere que esta sea una proposición.

en este sentido, también se podría usar una letra como p como identificador para cumplir la misma función de denotar. Típicamente, se usan identificadores como p, q, r, etc., que también pueden considerarse como nombres de variables que pueden asumir valores de verdad. Se habla de *variables proposicionales* o *sentenciales.*

Ya se ha dicho que una proposición es cierta o falsa, pero no ambas cosas. El sistema lógico que se va a usar para hablar de proposiciones debe tener, al menos, dos valores que se distingan para la verdad[2]. De hecho, se usan dos identificadores especiales, `true` y `false`, para representar lo que en la realidad es siempre verdadero y siempre falso, respectivamente[3].

Para simplificar la notación, ocasionalmente se usan los símbolos `1` y `0`, como taquigrafías de los identificadores `true` y `false` cuando se deban usar esas constantes en las llamadas tablas de verdad (cf. 2.1.4). Debe tenerse cuidado en entender que éstos son solo símbolos y que, por ejemplo, no se quiere operar aritméticamente los símbolos `1` y `0`. En realidad, como en cualquier notación matemática, lo que está mal es abusar de la notación, lo que correspondería a inferir de ella más significado del acordado.

2.1.1 Sintaxis y evaluación

Las proposiciones se conocen también como *expresiones booleanas*[4]. Enseguida se definirá la sintaxis de estas expresiones, que serán el lenguaje formal del sistema lógico conocido como lógica proposicional.

El alfabeto para definir el lenguaje de las expresiones booleanas tiene como símbolos `true`, `false`, identificadores para variables proposicionales, paréntesis y símbolos específicos para denotar relaciones entre proposiciones. Más que establecer un alfabeto preciso, se prefiere explicar la gramática y dejar que el alfabeto se infiera de la notación permitida para las expresiones bien formadas.

La siguiente gramática describe el lenguaje de la lógica proposicional que constituirá el sistema formal que aquí se define:

```
⟨expr-booleana⟩ →     true
                  |   false
                  |   ⟨variable⟩
                  |   ( ⟨expr-booleana⟩ ⟨opBin-booleano⟩ ⟨expr-booleana⟩ )
                  |   ¬( ⟨expr-booleana⟩ )

⟨opBin-booleano⟩→     ≡
                  |   ⇒
                  |   ⇐
                  |   ∧
                  |   ∨
                  |   ≢
⟨variable⟩      →  ⟨identificador⟩
```

---

[2] Hay lógicas donde se consideran más de dos valores de verdad. Aquí se usarán exactamente dos.

[3] El lenguaje formal del sistema lógico que se definirá usará un alfabeto con símbolos extraños al español. De esta manera se quiere separar el lenguaje natural del lenguaje formal.

[4] A partir de George Boole, pionero de la lógica simbólica, en el s. XIX.

Los llamados *operadores booleanos* sirven para construir expresiones booleanas a partir de otras ya construidas. Por ejemplo:

$$(p \wedge q),\ (p \Rightarrow q),\ \neg(p),\ \neg(p \equiv q),\ ((p \wedge q) \wedge \neg(p)),\ \ldots$$

Los *operadores binarios* que se han mencionado se usan como notación infija, i.e., como símbolos de una operación de dos operandos que se escriben a la izquierda y a la derecha del operador, de manera análoga a las operaciones binarios aritméticos, v.gr., 2+3.

En este mismo sentido, obsérvese que hay un *operador monario* o *unario*, cuyo símbolo es $\neg$, que se utiliza de manera prefija, i.e., a la izquierda de su operando. La analogía con la aritmética podría verse con la operación de cambio de signo, v.gr. -5.

2.1.1.1 Expresiones booleanas simplificadas

La gramática de la sección anterior permite definir proposiciones, sin ambigüedades. Sin embargo, también haciendo analogía con la aritmética, conviene preferir notaciones cuya legibilidad sea más evidente. Por ejemplo, se prefiere escribir 2+3*5 que (2+(3*5)).

La notación simplificada se consigue eliminando paréntesis. Esto se va a permitir de cuatro maneras:

- eliminando paréntesis:
  - los más externos;
  - los que rodean variables o constantes;
  - los duplicados para una misma expresión;
- estableciendo normas de precedencia de operadores;
- definiendo símbolos tachados;
- reconociendo asociatividad para ciertos operadores.

La eliminación de los paréntesis más externos no tiene problemas. Simplemente, se quitan dos símbolos cuya utilidad –para componer la proposición- no se considera. Tampoco hay problemas con quitar los paréntesis que rodeen variables o constantes, porque no hay estructura de composición en estos elementos. Y claro, si una expresión está rodeada por más de un par de paréntesis, basta quedarse con un par.

Las normas de *precedencia de operadores* reflejan reglas de evaluación, cuando se piensa en operadores y sus operandos. Así, en aritmética se aprende que la multiplicación tiene más precedencia que la suma, de modo que al evaluar 2+3*5 se empieza por evaluar 3*5 y, después, sumar 2 a ese resultado.

Las normas de precedencia se establecen con una lista de filas de operadores. Los operadores citados en una fila tienen menos precedencia que otros operadores citados en filas superiores, pero igual precedencia que los que aparecen en su misma fila. Para los operadores booleanos que se han mencionado, la precedencia se define así:

$$\neg$$
$$=$$
$$\vee \quad \wedge$$
$$\Rightarrow \quad \Leftarrow$$
$$\equiv$$

Los *símbolos tachados* son operadores tachados que se usan como abreviaturas para negaciones. Por ejemplo:

$b \not\equiv c$ es una abreviatura de $\neg(b \equiv c)$
$b \not\Rightarrow c$ es una abreviatura de $\neg(b \Rightarrow c)$
$b \not\Leftarrow c$ es una abreviatura de $\neg(b \Leftarrow c)$

Cuando se sabe que un operador es *asociativo* (como la suma, en la aritmética) los paréntesis se pueden omitir porque no importa el orden en que las operaciones se realicen. Entonces se prefiere 2+3+5 a 2+(3+5) y a (2+3)+5, ya que el resultado será igual siempre.

Por último, se puede asumir como convención que todos los operadores binarios no asociativos asocien a la izquierda. En este sentido, si ♦ es un operador binario, la expresión booleana

$$p \blacklozenge q \blacklozenge r$$

debe entenderse como una abreviatura de

$$(p \blacklozenge q) \blacklozenge r.$$

Si se quisiera la expresión

$$p \blacklozenge (q \blacklozenge r)$$

habría que dejarla así, con los paréntesis.

Como resumen de la simplificación de reglas, debe tenerse en cuenta que lo que se escriba debe ser una fórmula que corresponda a la simplificación de exactamente una fórmula bien formada del sistema lógico. Eso hará que la expresión pueda tener un significado único, no ambiguo. Y, cuando haya dudas, se pueden dejar paréntesis extra, aunque las normas de simplificación hubieran permitido algo más simple.

Ejemplo A: Simplificación de expresiones booleanas

La siguiente tabla muestra fórmulas correspondientes a expresiones booleanas, una versión simplificada y un comentario de la forma en que se realizó la simplificación.

	Fórmula según gramática	*Fórmula simplificada*	*Razones*
1	$\neg(p)$	$\neg p$	Paréntesis rodean variable
2	$((p \wedge q) \wedge \neg(p))$	$(p \wedge q) \wedge \neg p$	Paréntesis externos Paréntesis rodean variable
3	$((p \wedge q) \Rightarrow r)$	$p \wedge q \Rightarrow r$	Paréntesis externos $\wedge$ precede a $\Rightarrow$
4	$(p \wedge (q \Rightarrow r))$	$p \wedge (q \Rightarrow r)$	Paréntesis externos
5	$\neg((p \not\Rightarrow q))$	$\neg\neg(p \Rightarrow q)$	Paréntesis duplicados Símbolo tachado

En la medida en que se vea que algún operador binario sea asociativo se mostrará la forma simplificada de expresar las fórmulas correspondientes.

§

Ejemplo B: Expresiones ambiguas

La siguiente tabla muestra fórmulas ambiguas, en el sentido de que pueden entenderse de más de una manera. En la primera columna se muestran dichas fórmulas y, en la segunda y tercera, dos posibles maneras en que la fórmula de la primera columna podría entenderse.

	Fórmula ambigua	*Posibilidad 1*	*Posibilidad 2*
1	$p \wedge q \vee r$	$p \wedge (q \vee r)$	$(p \wedge q) \vee r$
2	$p \Rightarrow q \Leftarrow r$	$(p \Rightarrow q) \Leftarrow r$	$p \Rightarrow (q \Leftarrow r)$
3	$p \wedge q \wedge r$	$(p \wedge q) \wedge r$	$p \wedge (q \wedge r)$
4	$p \equiv q \equiv r$	$(p \equiv q) \equiv r$	$p \equiv (q \equiv r)$
5	$p \Rightarrow q \Rightarrow r$	$(p \Rightarrow q) \Rightarrow r$	$p \Rightarrow (q \Rightarrow r)$

Los ejemplos 1 y 2 son ambiguos, en el sentido de que hay más de una maneras de entenderlos. Se verá más adelante que las dos maneras son efectivamente diferentes (para esto hay que saber qué significan los símbolos).

Los ejemplos 3 y 4 son ambiguos como los dos primeros, pero también se verá que las dos maneras resultan equivalentes, porque los operadores van a resultar siendo *asociativos*.

El ejemplo 5 es ambiguo y tiene dos maneras de entenderse. Sin embargo, por la convención de que, en caso de ambigüedades como la que se muestra, el operador binario $\Rightarrow$ se entiende asociando a la izquierda, de modo que la fórmula debe entenderse como dice la posibilidad 1.

§

2.1.1.2 Árbol sintáctico de una fórmula

La construcción sintáctica de una fórmula da lugar a un *árbol sintáctico*, que describe la manera en que los operadores deben operar sus operandos. El árbol sintáctico asociado es único; si hay más de un árbol sintáctico asociable a una fórmula, esta situación corresponde exactamente a que la fórmula sea *ambigua*.

El árbol sintáctico se construye así:
(1) El árbol tiene como raíz el símbolo del último operador que se debe evaluar. También se le llama el *conectivo* u *operador principal* y corresponde a la última regla de la gramática que sirvió para construir la expresión. Si la expresión es una variable o un valor de verdad (no hay operadores), el árbol es, simplemente, un nodo raíz etiquetado con la variable o el valor en cuestión.
(2) Cuando el árbol tiene un símbolo de operador como raíz, los hijos del árbol son árboles sintácticos correspondientes a los operandos de la raíz.

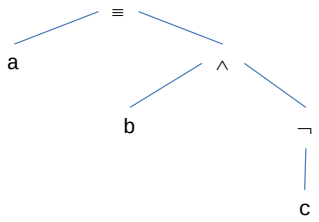
El árbol sintáctico de una expresión es una representación para ella, en el sentido de que, si se conoce el árbol se sabe cuál es la expresión. Un árbol es de gran utilidad para entender cómo se evalúa una fórmula.

Ejemplo A: Árboles sintácticos

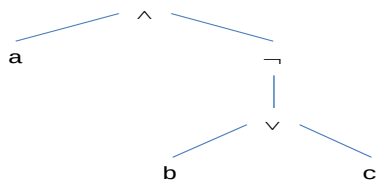
A continuación se muestran expresiones booleanas y árboles sintácticos correspondientes:

a :

$a \equiv b \wedge \neg c$:



$a \wedge \neg(b \vee c)$:



§

Ejercicios 2.1.1

1 Teniendo en cuenta la precedencia de operadores, construya árboles sintácticos para las siguientes expresiones:

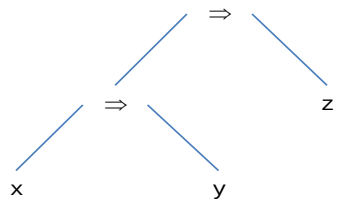
a $(x \wedge y) \vee \neg z$

b $x \wedge y \equiv x \vee y \equiv x \equiv y$

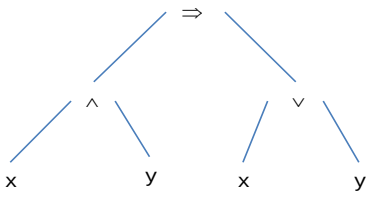
c $x \Rightarrow x \vee y$

2 Para cada uno de los siguientes árboles sintácticos, escriba la expresión booleana que corresponde:

a



b



2.1.2 Descripción de operadores booleanos

Como se indicó, los operadores booleanos permiten construir proposiciones a partir de otras proposiciones ya construidas. Éstas son los *argumentos* para el operador, el cual se interpretará después como una expresión que calcula un valor a partir de dichos argumentos.

Se pueden definir operadores booleanos con cualquier número de operandos. Muchas veces el operador se bautiza con un nombre que recuerda la operación que realiza ("y", "no", ...).

En matemáticas es usual denotar una fórmula en la que f, un operador n-ario que recibe los n argumentos $x_1, x_2, \ldots, x_n$, con una expresión como

$$f(x_1, x_2, \ldots, x_n).$$

En este caso, se dice que se usa una notación *prefija*: el operador precede los argumentos, y éstos se enmarcan en una lista parentizada.

A continuación se hace una relación de los operadores booleanos de 0, 1 y 2 argumentos. Además de entender su significado pretendido, se explicarán otras posibles notaciones para algunos de ellos.

2.1.2.1 Operadores 0-arios

Un operador `0`-ario no tiene argumentos. Entonces, es claro que siempre debe interpretarse de la misma manera, de modo que se pueden considerar *constantes*.

En las expresiones booleanas hay dos operadores `0`-arios, que se designan **F** y **T**. La siguiente tabla muestra el significado de estos operadores:

F	**T**
`0`	`1`

Aunque la notación para las expresiones que se pueden construir con estos operadores deberían ser **F**`()` y **T**`()`, se prefiere no usar los paréntesis. En otras palabras, **F** es el operador constante `false` y **T**, el operador constante `true`.

2.1.2.2 Operadores monarios

Un *operador booleano monario* (o *unario*) recibe un argumento de tipo booleano, i.e., `0` o `1`. Si se llama `x` este argumento, solo hay cuatro operadores booleanos posibles, cuyo valor se puede definir con la siguiente tabla (cada columna a la derecha de la del argumento `x` representa la definición de un operador posible):

x	**F**	**id**	**¬**	**T**
0	0	0	1	1
1	0	1	0	1

Las columnas marcadas **F** y **T** no son otra cosa que otra notación para los operadores monarios que tienen la misma denominación. Claramente son operadores constantes: el valor de su evaluación no depende de lo que valga el argumento.

La columna `id` es la del operador *identidad*. Este operador da como resultado el mismo valor de su argumento. Es decir, `id(false)=false` e `id(true)=true`.

La columna $\neg$ es operador de *negación*. Si el argumento `x` es `false`, $\neg$`x` es `true`, y viceversa.

La notación usada para $\neg$ no usa paréntesis sobre argumentos atómicos, v.gr., no se escribe $\neg$`(true)`, aunque tampoco sería problema hacerlo.

2.1.2.3 Operadores binarios

Un *operador booleano binario* `f` recibe dos argumentos booleanos. Si éstos son `x`, `y`, cada uno puede ser `false` o `true`, de modo que hay 4 posibilidades de valores de argumentos. Es decir, para entender `f` hay que conocer los valores asignados para `f(0,0)`, `f(0,1)`, `f(1,0)`, `f(1,1)`. Y como el valor de cada unos de éstos puede ser `false` o `true`, hay exactamente `16` ($= 2^4$) operadores binarios booleanos.

La tabla siguiente muestra cómo está definido cada operador binario. En general, se prefiere no usar notación prefija, sino

- *infija*, donde el operador queda entre los argumentos, como en $x \wedge y$, $x \vee y$, $x \Rightarrow y$
- *posfija*, donde el operador queda después de la lista de argumentos, como en $(x,y)_1$.

Las columnas se etiquetan con símbolos que denotan cada operador, aunque los símbolos mencionados no son –necesariamente- usados de manera estándar. Más aun, a veces hay más de un símbolo, porque se pueden usar otras notaciones.

x	y	F	$\wedge$	$\not\Rightarrow$	$\cdot_1$	$\not\Leftarrow$	$\cdot_2$	$\oplus$ $\neq$ $\not\equiv$	$\vee$	nor	= $\equiv$	$\neg_2$	$\Leftarrow$	$\neg_1$	$\Rightarrow$	nand	T
0	**0**	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	**1**	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	**0**	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	**1**	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

De nuevo, hay operadores constantes, cuya evaluación no depende del valor de los argumentos. También hay operaciones esencialmente monarias, como las marcadas como $\cdot_1$ y $\cdot_2$ (también llamadas *proyecciones* sobre el *primero* y el *segundo argumento*) que, en realidad, dependen solo de uno de los dos argumentos.

También hay operadores genuinamente binarios (v.gr., $\wedge$, $\vee$, $\Rightarrow$, $\equiv$), en el sentido de que el valor de una expresión construida con ellos depende verdaderamente de sus dos argumentos. Por ejemplo, el valor de $x \wedge y$ es `true` solo cuando x=y=`true`, o el de $x \Rightarrow y$ es `true`, a menos que x=`true` e y=`false`.

Las notaciones para los operadores `nor`, `nand`, $\not\equiv$, no son estándar. Por ejemplo, en ocasiones el operador $\not\equiv$ se suele representar con el símbolo $\oplus$.

Ejercicios 2.1.2

1 Un conjunto de operadores booleanos binarios se llama *completo* cuando todos los operadores binarios son expresables en términos de fórmulas que solo usan operadores del conjunto dado. Por ejemplo, el conjunto $\{\neg, \wedge, \vee, \Rightarrow\}$ es completo: para verificarlo hay que mostrar que todo operador booleano binario se puede mostrar equivalente a una expresión que solo utiliza los operadores mencionados. Para ilustrar este hecho

$x \equiv y \equiv (x \Rightarrow y) \wedge (y \Rightarrow x)$
$\neg x \equiv \neg x$
$x \Leftarrow y \equiv y \Rightarrow x$
etc.

a Compruebe que el conjunto $\{\wedge, \neg\}$ es completo.

b Compruebe que el conjunto $\{\wedge, \vee\}$ no es completo (no se pide una demostración; en cambio, argumente informalmente por qué debería ser así).

2 En el texto se vio que el número de operadores binarios es 16. Considere los operadores ternarios, i.e., operadores con 3 argumentos. ¿Cuántos operadores ternarios hay?

2.1.3 Semántica

Una expresión booleana se va a interpretar como un enunciado del lenguaje natural. Si la expresión es una constante, puede ser

- `true`, en cuyo caso se interpreta como un enunciado que siempre es verdadero;
- `false`, en cuyo caso se interpreta como un enunciado que siempre es falso.

Un siguiente caso corresponde a que la expresión booleana sea una variable p. La variable, como tal, puede representar un valor verdadero o un valor falso. En este sentido, el valor de la expresión booleana depende del caso que se quiera asumir para ella.

Para entender lo que puede ser el valor de una expresión booleana más compleja que una constante o una variable debe comprenderse primero lo que los operadores booleanos pretenden significar. Después, la semántica de las expresiones complejas se explicará en términos de la de sus partes.

2.1.3.1 Semántica de los operadores binarios

Las descripciones de los operadores de 2.1.2 incluyen las tablas que definen el valor que se entiende para las expresiones construidas con los diferentes operadores. Como la tabla de los operadores binarios incluye las de los constantes y las de los monarios, bastará referirse a la tabla de definiciones de 2.1.2.3.

Cada tabla pretende reflejar un significado 'natural' para el operador, en el sentido de que éste se lea de alguna manera que traduzca un conectivo del lenguaje natural ("y", "o", "entonces", "si y solo si", etc.).

Enseguida se explica cómo leer y entender algunos de los símbolos que denotan los operadores de esa tabla. Es usual, mientras se pueda, nombrar los operadores formales con alguna denominación en inglés o derivada de este idioma, para distinguirlos de sus significados pretendidos en español[5].

¬

Operador `NOT`
Corresponde a "no" en español. Es un operador monario.
¬p es verdadera cuando p es falsa; es falsa cuando p es verdadera.

∧

Operador `AND`
Corresponde a "y" en español. También se le dice 'conjunción'.
p ∧ q es verdadera solo cuando p y q lo son. En cualquier otro caso es falsa.

∨

Operador `OR`
Corresponde a "o" en español. También se le llama 'disyunción'.
p ∨ q es verdadera cuando p o q lo son; es falsa solo cuando los dos argumentos son falsos.
Se dice que es una disyunción inclusiva, porque es verdadera aun cuando los dos argumentos sean verdaderos.

≡

Operador `EQ` ("*is equal to*")
Corresponde al hecho de que los argumentos sean equivalentes en cuanto a su valor de verdad.
p ≡ q es verdadera si y solo si p y q son ambas verdaderas o ambas falsas.
El símbolo `'='` se usa con el mismo propósito. Sin embargo, más adelante este símbolo se usará para denotar la igualdad de elementos que no necesariamente representan valores booleanos.

⇒

Operador `IMP` ("*implies*")
p ⇒ q corresponde al hecho de que el argumento p, llamado el *antecedente*, implique el argumento q, llamado el *consecuente*.
p ⇒ q es verdadera en cualquier caso, excepto cuando p es verdadera y q es falsa. Nótese cómo esta definición se hace de acuerdo con la discusión que sobre implicar e inferir se hizo en 1.3.1.

⇐

Operador `CSQ` ("*is a consequence of*")
p ⇐ q corresponde exactamente a lo que represente q ⇒ p.

---

[5] Es decir, el español se está usando como lenguaje natural y el inglés como forma de leer el lenguaje formal correspondiente.

$\not\equiv$

Operador X0R ("exclusive or")
Tiene dos interpretaciones en español. La primera es entender que, como el operador es un símbolo de equivalencia tachado, $p \not\equiv q$ debe entenderse como se entiende $\neg(p \equiv q)$, es decir, es verdadero si y solo si p y q tienen diferentes valores de verdad. En este caso, una debe ser cierta y otra falsa, lo que lleva a la segunda interpretación: la no equivalencia corresponde a un "o exclusivo", i.e., una disyunción en la que los argumentos no son ambos verdaderos.

2.1.3.2 Semántica de expresiones booleanas

La semántica de una expresión booleana es el valor booleano que se entiende que la expresión representa. Como en el caso de una variable p, de la que no se puede afirmar que sea siempre cierta o siempre falsa, una expresión booleana puede ser cierta en ciertos casos y falsa en otros. Lo que sí será claro es que la verdad de una expresión depende de los valores de verdad que representen sus argumentos. Una *tabla de verdad* para una expresión e es una tabla que establece el valor de verdad de e en los diferentes casos que deben considerarse para las variables que en e aparecen.

Una tabla de verdad para la expresión booleana e tiene columnas para
- las variables que aparecen en e
- algunas subexpresiones booleanas que aparecen en la expresión (incluida la misma expresión e)

Una *subexpresión* e' (de la expresión e) es un argumento o una subexpresión de algún argumento. Al definir con tablas de verdad la semántica de las expresiones booleanas, se verá que es deseable incluir una columna por cada posible subexpresión. A veces el sentido común permite ahorrar en columnas (o bien, en considerar subexpresiones).

Las filas de la tabla, en las columnas de las variables, tienen valores booleanos. Cada fila corresponde a una manera diferente de definir los valores de las variables; cada una de estas maneras se denomina un *estado*. Si hay n variables, ya que cada una tiene dos posibles valores, hay 2^n filas o estados posibles.

En cada fila la tabla contiene, para cada subexpresión, su valor cuando las variables tienen los valores que están en la fila.

La columna correspondiente a la expresión e contiene los valores a los que evalúa en cada posible estado de las variables de las que depende.

Ejemplo A: Una tabla de verdad

Considérese la expresión $a \wedge \neg(b \vee c)$. Para hacer una tabla de verdad para ella, obsérvese que:
- Depende de 3 variables: a, b, c.
- Tiene 6 subexpresiones: a, b, c, $b \vee c$, $\neg(b \vee c)$, $a \wedge \neg(b \vee c)$

Las 3 variables dan lugar a 8 estados de evaluación posibles. La tabla se define así:

a	**b**	**c**	**b ∨ c**	**¬(b ∨ c)**	**a ∧ ¬(b ∨ c)**
0	**0**	**0**	0	1	**0**
0	**0**	**1**	1	0	**0**
0	**1**	**0**	1	0	**0**
0	**1**	**1**	1	0	**0**
1	**0**	**0**	0	1	**1**
1	**0**	**1**	1	0	**0**
1	**1**	**0**	1	0	**0**
1	**1**	**1**	1	0	**0**

Nótese que, para la determinación de las subexpresiones, es útil imaginar el árbol sintáctico de la expresión (cf. Ejemplo A de 2.1.1.2). Además, conviene listar las subexpresiones de izquierda a derecha en la tabla correspondiendo a las que están de abajo hacia arriba en el árbol sintáctico. De esta forma, los valores de las columnas de la tabla se pueden ir calculando en términos de los de las columnas más a la izquierda. El valor de la expresión como tal queda en la última columna a la derecha.

§

Una *tautología* e es una fórmula cuya tabla de verdad muestra que, en todos los estados posibles, evalúa a `true`. En otras palabras, es una fórmula que es verdadera en cualquier estado. Cuando así sucede, se dice también que e es *válida*. Nótese que para que una fórmula no sea una tautología basta mostrar que, en un estado dado, la evaluación correspondiente es `false`. Esto es lo que se denomina un *contraejemplo* para la validez de la expresión.

Análogamente, se dice que una fórmula es una *contradicción* si su tabla de verdad evalúa en todos los estados a `false`. Una contradicción es, por tanto, falsa en cualquier estado. También, para poder afirmar que una fórmula no sea una contradicción basta mostrar un contraejemplo, i.e., un estado en el que la fórmula evalúe a `true`.

Ejemplo B: Tautologías y contradicciones

(1) Claramente, la fórmula del Ejemplo A de 2.1.3 no es una tautología ni una contradicción. De hecho, solo es verdadera en el estado en el que `a=1`¸ `b=0`, `c=0`, de modo que este estado sirve de contraejemplo para evidenciar que la fórmula no es una contradicción. Así mismo, todos los demás estados sirven como contraejemplo para la afirmación de que la fórmula sea válida.

(2) Considérense las expresiones
(2a) $p \wedge q \equiv q \wedge p$
(2b) $p \wedge q \not\equiv q \wedge p$

Puesto que (2b) debe entenderse como $\neg$(2a), se hará una sola tabla de verdad para las dos fórmulas, donde se comprueba que (2a) es una tautología y (2b) es una contradicción:

p	**q**	**p ∧ q**	**q ∧ p**	**(2a)**	**(2b)**
0	**0**	0	0	**1**	**0**
0	**1**	0	0	**1**	**0**
1	**0**	0	0	**1**	**0**
1	**1**	1	1	**1**	**0**

Es fácil ver que la negación de una tautología es una contradicción y, además, que la negación de una contradicción es una tautología.

§

Si se quiere determinar si una expresión booleana es una tautología, la única manera de hacerlo que se tiene, por el momento, es elaborar la tabla de verdad, como en el ejemplo anterior. Lastimosamente, el procedimiento deja de ser práctico si el número de variables en la fórmula es demasiado grande o si la fórmula es demasiado larga. Lo primero hace que haya demasiadas filas: lo segundo, que haya demasiadas columnas[6].

6 En informática, un procedimiento de cálculo que dependa exponencialmente del número de datos de entrada se considera ineficiente e impráctico. En este sentido, el cálculo de tablas de verdad es intrínsecamente ineficiente, ya que debe establecerse el valor de la fórmula de interés en 2n estados posibles. Esto ya hace las cosas difíciles, pero lo son más, si se piensa que el número de subexpresiones sea m>1. Entonces, hay que calcular m*2n valores. El proceso de decisión, mediante tablas de verdad, para saber si una fórmula es o no tautología también requiere un número exponencial de pasos, porque se efectúa este cálculo y se verifica que la última columna siempre evalúe a true.

Ejercicios 2.1.3

Para cada una de las siguientes fórmulas, decida si es una tautología, una contradicción o ninguna de las anteriores. En cada caso, explique su respuesta.

1 $x \Rightarrow y \equiv \neg x \vee y$

2 $(x \Rightarrow y) \Rightarrow x$

3 $x \Rightarrow y \equiv x \vee \neg y$

4 $x \wedge (x \Rightarrow y) \wedge \neg y$

2.2 TRADUCCIÓN A Y DESDE LENGUAJE NATURAL

Cada expresión booleana encierra la intención de representar una aserción en lenguaje natural. Recuérdese que una aserción no es otra cosa que una frase que puede ser cierta o falsa. Así las cosas, puede haber frases más o menos complicadas y hay que buscar cómo representarlas de manera que el formalismo sirva para entenderlas y usarlas en deducciones, de la misma manera que las personas razonan.

Considérense las siguientes aserciones que podrían tener que ver con una realidad que hablara acerca de un personaje llamado Juan:

- "Juan tiene 21 años".
- "Juan estudia medicina".
- "Juan tiene 21 años y estudia medicina".

La sintaxis del lenguaje (cf. 2.1.1) permite definir expresiones booleanas con identificadores. También pueden usarse operadores binarios y el operador monario de la negación[7], a partir de expresiones booleanas ya definidas. ¿Cómo escoger la representación de estas aserciones?

Las expresiones booleanas más simples son las que corresponden a variables o identificadores. Por tanto, podrían usarse para las dos primeras aserciones, ya que son afirmaciones simples. Esto es así porque son aserciones que no contienen aserciones dentro de ellas:

`j21` : "Juan tiene 21 años"

`m` : "Juan estudia medicina".

La tercera aserción, en cambio, tiene claramente dos aserciones dentro de ella, conectadas por la conjunción "y". En el formalismo, una frase en lenguaje natural de la forma "aserción_1 y aserción_2" se representa con una fórmula como $\alpha \wedge \beta$, donde α es la representación de "aserción_1" y β la de "aserción_2". Para el ejemplo, se tendría que

$$j21 \wedge m$$

es la manera en que debería formalizarse la tercera aserción.

En este momento se ve la importancia de que el lenguaje formal cuente con operadores binarios que reflejen giros usuales en el lenguaje natural. Como se vio, algunos de los conectivos u operadores hacen una función gramatical parecida a las *conjunciones*, en español ("y", "ni", "o", "entonces"). Otros no tienen contraparte directa, pero se podrán traducir, como se verá.

La siguiente tabla – sin ser exhaustiva en los casos considerados - esquematiza cómo puede llevarse a cabo la traducción cuando las aserciones no son simples. La idea es que los argumentos en lenguaje natural se puedan representar de manera formal en el lenguaje del sistema. La columna de ejemplos es una muestra de frases en lenguaje natural que ilustran cada caso.

[7] En realidad, se puede permitir construir expresiones booleanas con cualquier operador como los de 2.1.2.3.

Lenguaje natural	*Ejemplos*	*Operador*	*Traducción*
y	Juan tiene 21 años y estudia medicina.	$\wedge$	`j21` $\wedge$ `m` donde `j21` : Juan tiene 21 años `m` : Juan estudia medicina
pero	Está lloviendo, pero hace sol.	$\wedge$	`v` $\wedge$ `s` donde `v` : está lloviendo `s` : hace sol
o (inclusivo)	Juan estudia medicina o biología.	$\vee$	`m` $\vee$ `b` donde `m` : Juan estudia medicina `b` : Juan estudia biología
o (exclusivo)	Este anillo es de oro o es de plata.	$\not\equiv$	`au` $\not\equiv$ `ag` donde `au` : el anillo es de oro `ag` : el anillo es de plata
no	Este anillo no es de oro	$\neg$	$\neg$`au` donde `au` : el anillo es de oro
no es el caso	No es el caso que Juan estudie biología.	$\neg$	$\neg$`b` donde `b` : Juan estudia biología
si … entonces …	Si hoy es domingo, mañana es lunes.	$\Rightarrow$	`d` $\Rightarrow$ `l` donde `d` : hoy es domingo `l` : mañana es lunes
… es suficiente para …	Que hoy sea domingo es suficiente para afirmar que mañana es lunes	$\Rightarrow$	`d` $\Rightarrow$ `l` donde `d` : hoy es domingo `l` : mañana es lunes
… es necesario para …	Que hoy sea domingo es necesario para que mañana sea lunes	$\Leftarrow$	`l` $\Leftarrow$ `d` donde `d` : hoy es domingo `l` : mañana es lunes
… si y sólo si …	`m` es par si y solo si `m+1` es impar	$\equiv$	`mp` $\equiv$ `im1` donde `mp` : `m` es par `im1` : `m+1` es impar
… es necesario y suficiente …	Para que el sistema tenga solución es necesario y suficiente que la matriz sea invertible	$\equiv$	`ss` $\equiv$ `minv` donde `ss` : es sistema tiene solución `minv` : la matriz es invertible

si … entonces … (cuando se definen conceptos)	Si un entero es divisible por 2 entonces es par	$\equiv$	$ed2 \equiv epar$ donde $ed2$: el entero es divisible por 2 $epar$: el entero es par

Cada caso referido en la tabla anterior da lugar a una traducción que, muchas veces no es tan literal. En el ejemplo de arriba, obsérvese que, si bien en lenguaje natural se podría decir "Juan tiene 21 años y Juan estudia medicina", se prefiere un giro como "Juan tiene 21 años y estudia medicina", omitiendo el sujeto en la segunda afirmación. En cualquier caso, debería ser claro que $j21 \wedge m$ modela adecuadamente lo que se quiere afirmar.

Una frase como "Juan tiene 21 años, pero estudia medicina" sigue traduciéndose como $j21 \wedge m$. Aquí se ve que el lenguaje de la lógica pierde en cierta clase de expresividad (como aquí, del matiz de que a uno no le parezca conveniente que alguien de 21 años estudie medicina) que, sin embargo, no importará para sacar deducciones.

Las frases que se traducen con el conectivo $\vee$ deben entenderse como una *disyunción inclusiva*, es decir, en la que alguna de las afirmaciones es verdadera, y ambas pueden serlo simultáneamente. Por otro lado, el conectivo $\not\equiv$ sirve para modelar una *disyunción exclusiva* de aserciones, i.e., alguno de los argumentos es verdadero, pero no ambos. Así, si "Juan estudia biología o medicina" se modela como $b \vee m$, se está afirmando que estudia alguna de las dos carreras, o incluso ambas. Pero si se modela como $b \not\equiv m$, se afirma que estudia una de las dos carreras pero sólo una de ellas.

Algunas frases en el lenguaje natural tienen una forma que podría, *a priori*, traducirse al formalismo de una manera distinta de la pretendida. Por ejemplo, si se dice: "Te tomas la sopa o no ves televisión" se podría traducir como $s \vee \neg t$, donde s modela "te tomas la sopa" y t significa "ves televisión". Sin embargo, si se piensa un poco más en el significado pretendido, sería mejor traducir como $\neg s \Rightarrow \neg t$[8]. Estas situaciones no son tan usuales pero, ya que pueden suceder, hacen pensar aun más que el proceso de traducción no debe ser literal.

Nótese que la estructura que se permite modelar con la lógica proposicional es la de frases cuya complejidad se establece con conjunciones o frases similares. No hay cuantificaciones en la lógica proposicional. Así una frase como "todo hombre es mortal" debería considerarse como una frase simple y representarse con una variable.

2.2.1 Argumentos correctos / incorrectos

En el uso corriente del lenguaje natural es usual encontrar argumentos en los que se enuncian varias frases como premisas que, entendidas como ciertas, conducen a poder afirmar una conclusión. Si las premisas y la conclusión se traducen a lógica proposicional, un tal argumento tiene la forma

$$p_1 \wedge p_2 \wedge \ldots \wedge p_s \Rightarrow c \qquad (A)$$

donde los p_i's son la traducción formal de las premisas y c la de la conclusión[9].

[8] Curiosamente, se verá que las dos posibles interpretaciones son equivalentes, aunque no lo parezcan en principio.

[9] En aras de no complicar la notación del ejemplo, habría que considerar diferentes formas de parentizar los elementos de las conjunciones del antecedente. Sin embargo, se puede comprobar que $\wedge$ es un operador asociativo, lo que garantiza que el orden de evaluación de las conjunciones no tiene complicación alguna y puede hacerse en cualquier orden.

Ahora se puede discutir si el argumento (A) es *correcto*, en el sentido de que, no importando los valores que se asignen a las variables, la implicación (A) sea una tautología.

Enseguida se presentan algunos ejemplos de esta clase de argumentos. Para mostrar si son o no correctos se procede de la siguiente manera:

a *Definición de variables*
Se definen variables de forma que
- representen proposiciones simples (atómicas, sin que otras proposiciones sean parte de ellas);
- los tiempos gramaticales no importen, en principio;
- se distinga qué variables aparecen en cada frase del argumento;
- (deseable) la frase represente una afirmación positiva o, al menos, con pocas negaciones (varias negaciones pueden inducir errores graves);
- se reconozca cuándo una frase ya ha sido representada por una variable.

b *Traducción a lógica proposicional*
Las frases del argumento se representan con expresiones booleanas, usando las variables y operadores booleanos para conectarlas. Debe tenerse en cuenta que
- cada oración debe corresponder a una expresión booleana;
- frases separadas por puntos o puntos y comas se unen mediante conjunciones ($\wedge$). Es bueno dar un identificador a cada frase, para saber también en qué expresión se traduce;
- la última frase está conectada por un "entonces", "luego", "por tanto", etc. a la conjunción de las anteriores. Es decir hay un operador $\Rightarrow$ conectándolas;
- se llega a una expresión booleana correspondiente al argumento, de la forma

$$p_1 \wedge p_2 \wedge \ldots \wedge p_s \Rightarrow c \ .$$

c *Tabla de verdad*
La corrección del argumento se decide según una tabla de verdad para su expresión correspondiente de acuerdo con el paso anterior:
- si esta expresión es una tautología, el argumento es correcto, ya que es verdadero en cualquier caso que se pueda dar;
- si la expresión no es una tautología, el argumento es incorrecto. Cada estado en el que su evaluación sea falsa corresponde a un contraejemplo para la corrección.

El método anterior es *completo*, en el sentido de que de esta manera se puede decidir la corrección de cualquier argumentación. Sin embargo, tiene problemas de factibilidad, porque puede resultar demasiado dispendioso para ser llevado a la práctica en casos complejos.

Ejemplo A: Argumento correcto

Comprobar si el siguiente razonamiento es correcto:
Si María no tiene dientes, Dios le da pan.
Dios no le dio pan a María o ella tiene dientes.
Entonces, María tiene dientes.

a *Definición de variables*
En el argumento se pueden definir dos variables, d y p:
d : María tiene dientes
p : Dios le da pan a María

b *Traducción a lógica proposicional*
(P1) $\neg d \Rightarrow p$ // Si María no tiene dientes, Dios le da pan.
(P2) $\neg p \vee d$ // Dios no le dio pan a María o ella tiene dientes.
(C) d // María tiene dientes.

La expresión correspondiente al argumento es

$$P1 \wedge P2 \Rightarrow C$$

c Tabla de verdad

C		[1]	P1	[2]	P2	[3]	
d	**p**	$\neg$**d**	**[1] $\Rightarrow$ p**	$\neg$**p**	**[2] $\vee$ d**	**P1 $\wedge$ P2**	**[3] $\Rightarrow$ C**
0	**0**	1	0	1	1	0	**1**
0	**1**	1	1	0	0	0	**1**
1	**0**	0	1	1	1	1	**1**
1	**1**	0	1	0	1	1	**1**

La afirmación es válida (ver última columna). Entonces, el argumento es correcto.

§

Ejemplo B: Argumento incorrecto

Comprobar si el siguiente razonamiento es correcto:
Si Dios no le da pan a María, ella tiene dientes.
Dios le da pan a María.
Entonces María no tiene dientes.

a Definición de variables
d : María tiene dientes
p : Dios le da pan a María

b Traducción a lógica proposicional
(E1) $\neg p \Rightarrow d$ // Si Dios no le da pan a María, ella tiene dientes.
(E2) p // Dios le da pan a María
(C) $\neg d$ // María no tiene dientes

Probar:

$$E1 \wedge E2 \Rightarrow C$$

Tabla de verdad

E2		[1]	E1	[3]	C	
d	**p**	$\neg$**p**	**[1] $\Rightarrow$ d**	**E1 $\wedge$ E2**	$\neg$**d**	**[3] $\Rightarrow$ C**
0	0	1	0	0	1	**1**
0	1	0	1	1	1	**1**
1	0	1	1	0	0	**1**
1	1	0	1	1	0	**0**

El argumento no es correcto. Como no es una tautología, hay al menos un caso de la conclusión que es falso, i.e., un escenario descrito por los valores de verdad de las variables, lo que constituye un contraejemplo. Para este argumento, cuando p y d son `true`. Es decir, se ve que si Dios le dan pan a María y María tiene dientes, no puede ser verdad que María no tenga dientes.

§

Ejemplo C: ¿Existe Supermán?[10]

Probar que el siguiente razonamiento es correcto:

Si Supermán fuera capaz y quisiera prevenir el mal, él lo prevendría.
Si Supermán fuera incapaz de prevenir el mal, sería impotente; si él no quisiera prevenir el mal, sería malévolo.
Supermán no previene el mal.
Si Supermán existe, no es impotente ni malévolo.
Entonces,
Supermán no existe.

a Definición de variables

F0 Si Supermán fuera capaz y quisiera prevenir el mal, él lo prevendría.
a : Supermán es capaz de prevenir el mal
w : Supermán quiere prevenir el mal
p : Supermán previene el mal

F1 Si Supermán fuera incapaz de prevenir el mal, sería impotente; si él no quisiera prevenir el mal, sería malévolo.
i : Supermán es impotente
m : Supermán es malévolo

F2 Supermán no previene el mal.

F3 Si Supermán existe, no es impotente ni malévolo.
e : Supermán existe

F4 Supermán no existe.

b Traducción a lógica proposicional

(F0) $a \wedge w \Rightarrow p$
(F1) $(\neg a \Rightarrow i) \wedge (\neg w \Rightarrow m)$
(F2) $\neg p$
(F3) $e \Rightarrow \neg i \wedge \neg m$
(F4) $\neg e$

A demostrar:

$$F0 \wedge F1 \wedge F2 \wedge F3 \Rightarrow F4$$

c Tabla de verdad

Obsérvese que hay 6 variables (a,w,p,i,m,e). Por tanto, una tabla de verdad debe tener $2^6 = 64$ estados posibles (renglones en la tabla),

El número de columnas se puede estimar por el número de operadores en las frases. Hay 13 operadores en F0,…,F4 y 4 más en el argumento como tal (lo que se debe demostrar).

En este ejemplo, hay que llenar una tabla de 64*17 = 1088 casillas …

§

[10] Este ejemplo es tomado de [Gri1993] que a su vez lo refiere a textos de R. Backhouse , A. Foster y G. Shute. Originalmente es un argumento acerca de la inexistencia de Dios.

Ejercicios 2.2.1

1 En cada una de las siguientes frases, identifique proposiciones simples usando variables o símbolos proposicionales. Asegúrese de buscar significados afirmados de forma positiva (evitando negaciones). Luego escriba la expresión booleana que describa la frase original. Finalmente, escriba la contrarrecíproca de la expresión y como frase en español de la forma más natural que pueda.

a Si hoy es martes, entonces hoy no es miércoles.
b Si llueve, entonces no iré a hacer mercado.
c Me quedaré solo si tú te vas.
d Me quedaré sólo si tú te vas.
e No puedo terminar la tarea si no entiendo la demostración.
f Está lloviendo, así que no puedo ir al pueblo.
g No ha nevado, así que no puedo esquiar.

2 Verifique la corrección de los siguientes argumentos:

a Si el Sr. Suárez o la Sra. Suárez ganan más de 10 salarios mínimos legales mensuales, la familia Suárez puede pasar las vacaciones en Miami. Puesto que yo sé que, o el Sr. Suárez o su esposa ganan más de 10 salarios mínimos mensuales, concluyo que la familia puede pasar las vacaciones en Miami.

b Si no lloviera o si no hubiera neblina, ayer se habría llevado a cabo la regata. Cuando hay regata, se entrega un trofeo. El trofeo no fue entregado. Por lo tanto, sabemos que llovió.

c El crimen fue cometido por X o por Y. X estaba fuera del pueblo cuando el crimen fue cometido. Si X estaba fuera del pueblo, no pudo haber estado en la escena del crimen. Si X no estaba en la escena del crimen, no pudo haber cometido el crimen. Así que el crimen fue cometido por Y.

d Si hoy es martes, tendré un examen de lógica o un examen de economía. Si el profesor de economía se enferma, entonces no tendré examen de economía. Hoy es martes y mi profesor de economía está enfermo. Por lo tanto tendré un examen de lógica.

3 Muestre que el siguiente razonamiento no es correcto:
Si llueve Juan se queda en casa. Si Juan tiene tareas se queda en casa. Juan se quedó en casa.
Entonces, Juan tiene tareas o está lloviendo.

2.3 REPRESENTACIÓN COMPUTACIONAL

El Ejemplo C de 2.2 ilustra cómo un problema de un tamaño no muy grande puede ser muy dispendioso y costoso de resolver. Una salida que debe considerarse es la posibilidad de usar un computador para descubrir la solución. En otras palabras, representar computacionalmente las variables de la fórmula en cuestión y, sistemáticamente, construir una tabla de verdad correspondiente a la expresión booleana. Como antes, el razonamiento es correcto si y solo si la fórmula resulta ser una tautología.

2.3.1 Hojas de cálculo

Las hojas de cálculo, v.gr. MS Excel, manejan de manera primitiva tablas de valores. Una tabla de verdad se puede representar fácilmente en una hoja de cálculo. Esto es factible, sobre todo, si el número de variables no crece más allá de los límites de la memoria RAM de que se dispone.

El Apéndice A muestra cómo puede llevarse a cabo esta traducción de manera sistemática en MS Excel.

Ejercicios 2.3.1

1 Resuelva con ayuda de tablas de verdad representadas en Excel los problemas planteados en los Ejemplos A, B y C de 2.2.1

2.3.2 Lenguajes de programación

Los lenguajes de programación tienen, usualmente, forma de representar expresiones booleanas de manera primitiva. Esto quiere decir que resulta fácil representar expresiones booleanas simples con variables booleanas del lenguaje y que hay conectivos u operadores booleanos que permiten representar las operaciones booleanas complejas.

A continuación se muestra la correspondencia entre expresiones booleanas y el lenguaje Java, por dar un ejemplo de un lenguaje de programación muy conocido y utilizado:

Elemento	**Lógica**	**Java**
Variable booleana	p	**boolean** p;
Valor verdadero	true	true
Valor falso	false	false
Negación	$\neg$p	!p
Conjunción	p $\wedge$ q	p && q
Disyunción	p $\vee$ q	p || q

De esta manera, resulta también fácil representar tablas de verdad en un lenguaje como Java.

Por último, en Java las operaciones booleanas se pueden extender a cadenas de bits. Si las cadenas de bits pueden representar valores binarios, las operaciones aritméticas se pueden definir a partir de operaciones lógicas sobre estas cadenas.

Ejemplo A: Aritmética y operadores booleanos

Como ilustración de la forma en que la aritmética se puede llevar a cabo en términos de operadores lógicos se mostrará cómo puede definirse un *sumador binario*, i.e., un procedimiento para sumar cadenas de bits que representen números binarios.

Para simplificar, se va a mostrar cómo pueden sumarse números binarios de solo 3 bits. Supóngase que se tienen dos números binarios, representados con las cadenas de 3 bits así:

M = a2 a1 a0
N = b2 b1 b0

donde las las cifras menos significativas se entienden a la derecha. Así, si a2=1, a1=1, a0=0, la cadena M representa el valor binario 110, correspondiente al decimal 6.

¿Cómo explicar cómo se suma en binario? En general, si se suman dos binarios de r bits, el resultado puede tener un máximo de r+1 bits. En este caso, el resultado de sumar dos números de 3 bits es, cuando más, un número de 4 bits, que puede definirse como la cadena

R = x3 x2 x1 x0

Explicar la suma en binario significa definir cómo se calculan los bits de la cadena R = x3 x2 x1 x0. Para esto se procede así:

x0 = a0 $\oplus$ b0

Sea c1 = a0 $\wedge$ b0. Entonces:

x1 = a1 $\oplus$ b1 $\oplus$ c1

Sea c2 = (a1 ∧ b1) ∨ (a1 ∧ c1) ∨ (b1 ∧ c1). Entonces:

x2 = a2 ⊕ b2 ⊕ c2

x3 = (a2 ∧ b2) ∨ (a2 ∧ c2) ∨ (b2 ∧ c2)

Los valores c1 y c2 corresponden a "llevar 1", cuando los números se suman manualmente.

§

Ejercicios 2.3.2

1 Resuelva con ayuda de tablas de verdad representadas en Java los problemas planteados en los Ejemplos A, B y C de 2.2.1.

2 Desarrolle un programa Java que reciba una expresión booleana y decida si ésta es una fórmula válida.

2.4 DEDUCCIÓN

Como se vio, la semántica de las expresiones de lógica proposicional se define en términos de tablas de verdad. En la práctica, es necesario pensar si una fórmula es válida y en qué casos lo es. Entonces, se ve que el método de construir tablas de verdad puede resultar inviable, si se tiene en cuenta que el número de filas de la tabla depende exponencialmente del número de variables y que el número de columnas depende del número de subexpresiones de la fórmula. El método de establecer la validez de una fórmula resulta muy susceptible al error humano y, cuando se piensa en ayudas computacionales, costoso en espacio y en tiempo.

Hay otra forma de establecer la validez de una fórmula, basada en un *aparato deductivo* como los ya explicados en el Cap. 1. La idea con este método alternativo es razonar formalmente a imagen y semejanza de cómo un humano podría hacerlo. En principio, se parte de aceptar algunas fórmulas, llamadas *axiomas*, como expresiones válidas. Después, mediante *reglas de inferencia*, efectuar cadenas de deducciones formales con las que se establece la validez de nuevas fórmulas. Más adelante se puede pensar en esquemas alternativos de deducción, basados en esquemas ya aceptados como correctos.

Recuérdese que se llamarán *teoremas* a los axiomas o a las fórmulas válidas que puedan deducirse de teoremas ya demostrados, usando el aparato deductivo. Las deducciones formales deben simular la forma en que se razona. Se usará la siguiente notación:

$$\vdash \alpha$$

para significar que la fórmula α es un teorema.

2.4.1 Axiomas

Los *axiomas* en el aparato deductivo de un sistema lógico son fórmulas válidas a partir de las cuales se pueden deducir nuevas fórmulas válidas.

Cualquier fórmula válida puede ser un axioma. No hay que demostrar un axioma, pero sí hay que justificar por qué es una fórmula que vale en cualquier caso. En lógica proposicional la justificación de que una fórmula sea un axioma es muy sencilla de proveer: la tabla de verdad de la fórmula debe indicar que sí es una fórmula válida.

La tabla siguiente muestra los axiomas que se usarán aquí como base del aparato deductivo con el que se demostrará la validez de fórmulas[11]. Cada axioma tiene una frase que sirve de mnemotecnia para nombrarlo. Algunas de estas frases no identifican unívocamente a un axioma (un mismo nombre es compartido por varias fórmulas), pero se verá que esto no causa problemas a la hora de escribir demostraciones o de entenderlas.

La tabla agrupa, con algún orden, axiomas que tengan que ver con los diferentes operadores o conectivos lógicos. Cada uno de ellos puede justificarse comprobando su validez al construir la tabla de verdad correspondiente.

Debe entenderse que, para usar un axioma, basta remplazar cualquier variable que aparezca en él por una expresión booleana cualquiera, lo que da como resultado una fórmula válida: si el axioma es verdadero en cualquier estado, también lo es en el que se defina para la expresión que remplaza la variable.

Identificador	***Axioma***
true	`true`
≡- Asociatividad	`((p ≡ q) ≡ r) ≡ (p ≡ (q ≡ r))`
≡- Conmutatividad	`p ≡ q ≡ q ≡ p`
≡- Identidad	`true ≡ p ≡ p`
Def false	`false ≡ ¬true`
Neg ≡	`¬(p ≡ q) ≡ ¬p ≡ q`
Def ≠	`x ≠ y  ≡ ¬(x ≡ y)`
Doble negación	`¬¬p ≡ p`
∨-Identidad	`p ∨ false ≡ p`
∨-Conmutatividad	`p ∨ q ≡ q ∨ p`
∨-Dominancia	`p ∨ true ≡ true`
∨-Asociatividad	`p ∨ (q ∨ r) ≡ (p ∨ q) ∨ r`
∨-Idempotencia	`p ∨ p ≡ p`
Medio excluído	`p ∨ ¬p ≡ true`
∧-Identidad	`p ∧ true ≡ p`
∧-Dominancia	`p ∧ false ≡ false`
∧-Conmutatividad	`p ∧ q ≡ q ∧ p`
∧-Asociatividad	`p ∧ (q ∧ r) ≡ (p ∧ q) ∧ r`
∧-Idempotencia	`p ∧ p ≡ p`
Contradicción	`p ∧ ¬p ≡ false`
Distributividad ∨/≡	`p ∨ (q ≡ r) ≡ p ∨ q ≡ p ∨ r`
Distributividad ∧/∨	`p ∧ (q ∨ r) ≡ (p ∧ q) ∨ (p ∧ r)`
Distributividad ∨/∧	`p ∨ (q ∧ r) ≡ (p ∨ q) ∧ (p ∨ r)`
De Morgan	`¬(x ∨ y) ≡ ¬x ∧ ¬y`
De Morgan	`¬(x ∧ y) ≡ ¬x ∨ ¬y`
Absorción	`p ∨ (p ∧ q) ≡ p`
Absorción	`p ∧ (p ∨ q) ≡ p`

[11] Obsérvese que el operador de equivalencia ≡ está presente en todos estos axiomas. Esta peculiaridad va a permitir que las deducciones se hagan, en principio, respetando equivalencias, lo que dará lugar a una lógica conocida como *ecuacional*.

Absorción $\neg$	$p \vee (\neg p \wedge q) \equiv p \vee q$
Absorción $\neg$	$p \wedge (\neg p \vee q) \equiv p \wedge q$
Def $\Rightarrow$	$p \Rightarrow q \equiv \neg p \vee q$
Contrarrecíproca	$p \Rightarrow q \equiv \neg q \Rightarrow \neg p$
Distributividad $\Rightarrow/\wedge$	$p \Rightarrow (q \wedge r) \equiv (p \Rightarrow q) \wedge (p \Rightarrow r)$
$\wedge\Rightarrow/\Rightarrow\Rightarrow$	$p \wedge q \Rightarrow r \equiv p \Rightarrow (q \Rightarrow r)$
Def $\equiv$	$p \equiv q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$
Def $\equiv$	$p \equiv q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$

Los axiomas de *asociatividad* de $\vee$, $\wedge$ y $\equiv$ permiten simplificar expresiones omitiendo paréntesis innecesarios. Así, se entiende que

$\alpha \equiv \beta \equiv \gamma$ abrevia $((\alpha \equiv \beta) \equiv \gamma)$ o $(\alpha \equiv (\beta \equiv \gamma))$
$\alpha \wedge \beta \wedge \gamma$ abrevia $((\alpha \wedge \beta) \wedge \gamma)$ o $(\alpha \wedge (\beta \wedge \gamma))$
$\alpha \vee \beta \vee \gamma$ abrevia $((\alpha \vee \beta) \vee \gamma)$ o $(\alpha \vee (\beta \vee \gamma))$.

La propiedad de *conmutatividad* también es llamada *simetría*.

Ejercicios 2.4.1

1 Decida si cada una de las siguientes expresiones puede ser un axioma:
a $p \wedge (q \Rightarrow p) \equiv p$
b $(p \Rightarrow q) \wedge (q \Rightarrow r) \wedge (r \Rightarrow s) \equiv p \Rightarrow s$

2 Compruebe que $\vee$ es el único operador binario que cumple los axiomas $\vee$-Identidad, $\vee$-Conmutatividad, $\vee$-Dominancia, $\vee$-Asociatividad, $\vee$-Idempotencia y Medio excluído. Es decir, estos axiomas *caracterizan* el comportamiento del operador $\vee$.

2.4.2 Reglas de inferencia

Las reglas de inferencia del cálculo proposicional son métodos usados para deducir teoremas a partir de otros teoremas. Como los teoremas son fórmulas válidas, y las reglas tienen la forma

Identificador : $\dfrac{H_1, \ldots, H_k}{C}$ | Condiciones de aplicación

se espera que, si las hipótesis $H_1, \ldots, H_k$ son válidas, la conclusión C también lo sea. Una forma de escribir la regla, en una línea, es también aceptable (cuando no hay condiciones de aplicación):

$$H_1, \ldots, H_k \vdash C$$

Como los axiomas, las reglas de inferencia no se demuestran, pero sí se justifican. Una regla de inferencia es *correcta* cuando, si se suponen válidas las hipótesis H_1, …, H_k, se puede concluir la validez de la conclusión C. En otras palabras, la regla es correcta si

$$H_1 \wedge H_2 \wedge \ldots \wedge H_k \Rightarrow C$$

es una tautología. Obsérvese que esta es la misma justificación que se dio en 2.2.1 para reconocer un argumento correcto.

En las hipótesis de una regla puede haber variables que pueden remplazarse por fórmulas. *Aplicar una regla* consiste en establecer, para algunas variables que aparezcan en las hipótesis, expresiones que las

sustituyan de manera como convenga. Después, esa misma sustitución de variables por expresiones se realiza sobre la conclusión (teniendo en cuenta las condiciones que puedan restringir la aplicación).

La notación

$$E[x := p]$$

se utiliza para significar que en la expresión E se remplazan las apariciones de la variable x por la expresión p. Es decir, [x:= p] denota una sustitución que, en este caso, se aplica a E. En un caso general, se usa la notación

$$E[x1,x2,\ldots,xn := p1,p2,\ldots,pn]$$

para establecer que, en la expresión E, las variables x1,x2,…,xn se sustituyen por las expresiones p1,p2,…,pn, de manera simultánea[12].

La acción de aplicar una regla es lo que se llama *deducir*. Así, de teoremas se deducen teoremas.

2.4.2.1 Reglas de inferencia ecuacional

Las equivalencias sobre ≡ (cf. 2.4.1) justifican la introducción de reglas de inferencia correctas como:

Reflexividad :
$$\frac{p \equiv p}{true}$$

Simetría :
$$\frac{p \equiv q}{q \equiv p}$$

Transitividad :
$$\frac{p \equiv q,\ q \equiv r}{p \equiv r}$$

Por ejemplo, como regla de inferencia, la reflexividad de ≡ sirve para asegurarse de que si vale p ≡ p, también debe valer true. Estas reglas tienen la peculiaridad de que se pueden leer como reglas en sentido contrario: de la conclusión se pueden deducir las hipótesis (en la transitividad, usada al contrario, habría que inventarse un q equivalente a p, y éste sería equivalente a r, cuando p ≡ r fuera válido).

Una *regla de inferencia ecuacional* permite concluir una equivalencia en su conclusión. Este es el caso de las reglas de simetría y de transitividad mencionadas. El que estas reglas hagan parte del cálculo da lugar a esquemas de presentación de *pruebas* (*demostraciones*) de la forma (cf. 1.2.2):

E_0
≡ ⟨razón 1⟩
E_1
≡ ⟨razón 2⟩
…
≡ ⟨razón k⟩
E_k

[12] Esta notación para sustituciones de variables tiene problemas en casos extremos, v.gr., cuando algunas de las xi's estuviera repetida y fuera sustituida por dos expresiones booleanas diferentes. Esto puede evitarse en la definición de la notación, pero es más sencillo –como introducción al tema- no usar variables repetidas y evitar conflictos donde el orden de sustitución de variables importe. Siempre se pueden hacer varios pasos para expresar lo que se quiere decir.

La ⟨razón i⟩ que se escribe entre las líneas i-1 e i, explica qué regla de inferencia justifica la equivalencia $E_{i-1} \equiv E_i$. Nótese que, si todas las razones son justificadas (¡son correctas!), todas las expresiones E_0, …, E_k son equivalentes entre sí. También se usa el mismo esquema con operadores = en lugar de los operadores ≡.

La identidad, la simetría y la transitividad de la equivalencia ≡, avalan el esquema como forma de presentación de una prueba. Una manera informal, pero aceptable, de usar esta clase de razones, es señalar algo como SAP, como sigla de "simetría, asociatividad o parentización". Si se usa esta taquigrafía, se debe entender que la fórmula resultado es equivalente a la inicial después de aplicar reglas de esta clase en los conectivos que haga falta y que –se sabe- tienen estas propiedades[13].

Hay una diferencia importante en las deducciones cuando se usan axiomas y reglas de equivalencia ecuacionales, frente al uso de axiomas y reglas que no lo sean. De acuerdo con lo que se definió como prueba en el Cap. 1, al hablar de sistemas formales y de sistemas lógicos, en principio, una prueba debe entenderse como una cadena de implicaciones de la forma[14]

E_0
⇒ ⟨razón 1⟩
E_1
⇒ ⟨razón 2⟩
…
⇒ ⟨razón k⟩
E_k

Con una demostración así se podría probar que $E_0 \Rightarrow E_k$. Pero, claramente, esta *no es una prueba de* $E_k \Rightarrow E_0$. El axioma

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

y el significado de la conjunción ∧, establecen que la equivalencia de p y q equivale, a su vez, a la implicación de q desde p y viceversa. Es decir, si lo que se quiere probar es la equivalencia de dos proposiciones p y q, una demostración en un sistema lógico cuyos axiomas y reglas de inferencia sigan los lineamientos del Cap. 1 debe constar, en rigor, de una de prueba en un sentido de la implicación y de otra en el sentido contrario.

Si se trata de probar equivalencias, si el cálculo deductivo es ecuacional (en axiomas y en reglas), es usual que se pueda hacer demostraciones más cortas que en cálculos no ecuacionales (cf. 2.4.3).

2.4.2.2 Sustitución

La regla de *Sustitución* se refiere al hecho de que se conozca que una expresión proposicional sea verdadera, que puede o no depender de una variable proposicional x. Si una aparición de x se remplaza por *cualquier proposición* p, la fórmula resultante también debe ser verdadera.

Sustitución :
$$\frac{E}{E[x:=p]}$$
p : proposición
E : expresión proposicional

Por ejemplo, si se sabe que es cierto que

$$x \vee (\neg x \wedge true)$$

[13] Los cambios en la parentización son una consecuencia de las reglas de precedencia de operadores que se adopten.

[14] Se requiere justificar una transitividad de la implicación.

entonces, si en esta expresión la variable x se remplaza por una proposición cualquiera, v.gr., $q \Rightarrow \neg s$, también debe ser cierto que

$$(q \Rightarrow \neg s) \vee (\neg(q \Rightarrow \neg s) \wedge true)$$

ya que (recuérdese el significado de aplicar una sustitución);

$(x \vee (\neg x \wedge true))[x:= q \Rightarrow \neg s]$

$\equiv$

$(q \Rightarrow \neg s) \vee (\neg(q \Rightarrow \neg s) \wedge true)$

2.4.2.3 Regla de Leibniz

La regla de Leibniz refleja la legalidad de "cambiar iguales por iguales".

Leibniz :

$$\frac{p \equiv q}{E[p] \equiv E[q]}$$

p,q : proposiciones
E[x]: expresión sobre x

Detrás de la regla de Leibniz está la noción de que, cuando se sabe que dos cosas son iguales, su valor y su uso son indistinguibles. Hablando de expresiones booleanas, si dos expresiones booleanas son equivalentes (i.e., son verdaderas o son falsas simultáneamente), una de ellas se puede cambiar por la otra en cualquier otra expresión booleana, sin que el valor de verdad de esa expresión se vea alterado.

Ejercicios 2.4.2

1 La notación de sustituciones es, en general de la forma

$$[x:= p]$$

para establecer que en el vector de variables $x = \langle x_1, x_2, \ldots, x_n \rangle$ se sustituyen por las del vector de expresiones $p = \langle p_1, p_2, \ldots, p_n \rangle$, de manera simultánea. En lugar de variables simples también se pueden usar referencias a arreglos de variables, por ejemplo:

$$[b[j+1],i:= b[j]+1,i+1]]$$

No debe usarse cuando los elementos de la izquierda se repitan . Por ejemplo, en

$$[x,x:= 1,2]$$

o, cuando $i=j$, en

$$[b[j],b[i]:= 1,2]$$

el significado es ambiguo. Estas situaciones deben evitarse.

Por último, en la aplicación de sustituciones se da prioridad a las que están al lado de la expresión que se sustituye, es decir:

$$E[x:= p][y:= q] = (E[x:= p])[y:= q].$$

En los siguientes ejercicios calcule la expresión resultante después de aplicar la sustitución. Si la buena definición requiere de alguna condición especial, indíquela.

a $x[x:= a+3]$
b $x+y*x[x:= a+3]$
c $(x+y*x)[x:= a+3]$
d $x+y*x[x,y:= y,x]$
e $(x+y*x)[x,y:= y,x]$
f $(b[i]+5*b[j])[i,b[i],b[j]:= i+1,2,3]$
g $(b[i]+5*b[j])[i:= i+1][b[i]:= 2][b[j]:= 3]$

2 Suponga una variante de la Regla de Leibniz:

Leibniz1 : $\dfrac{Z \equiv X, Z \equiv Y}{E[z:=X] \equiv E[z:=Y]}$ | X,Y,Z : proposiciones; E[x]: expresión sobre x

Muestre que si en el cálculo se remplaza la regla Leibniz por la regla Leibniz1 se obtienen exactamente las mismas deducciones.

2.4.3 Demostración con equivalencias

Supóngase que se quiere demostrar

$$\vdash \alpha$$

i.e., probar que una expresión α es un teorema. Con el sistema lógico ecuacional que se ha presentado se seguirá el siguiente esquema de demostración:

Teo : α
Dem:
E_0
= ⟨razón 1⟩
E_1
= ⟨razón 2⟩
…
= ⟨razón k⟩
E_k

□

La demostración anterior es *correcta* (o sea, α es un teorema) si se da alguna de las siguientes situaciones:

i α es `true`
ii α es E_0 y E_k es un teorema
iii α es E_k y E_0 es un teorema
iii α es $E_0 \equiv E_k$

Una expresión α en cualquiera de los anteriores casos es una *expresión ecuacional*. Si se demuestra, se tendrá un *teorema ecuacional*. Nótese que, si este es el caso, α es equivalente a un teorema (que puede ser `true`) o α es expresable como la equivalencia de dos subfórmulas que el esquema, precisamente, muestra que lo son.

Una *razón* es una explicación textual que debe citar una regla de inferencia o un teorema ecuacional ya demostrado, que justifica la equivalencia de las dos líneas que la enmarcan. Así, la razón `i` justifica la equivalencia $E_{i-1} \equiv E_i$, `i=1,…,k`. Eventualmente se puede incluir una razón `0`, anotada antes de E_0, que justifique empezar con E_0, cuando éste corresponde a un teorema demostrado.

La información de la razón `i` queda completa indicando
- identificador de la regla de inferencia o del teorema que se aplica
- enunciado de la regla de inferencia o del teorema que se aplica
- parte de la fórmula E_{i-1} sobre la que se aplica el teorema o la regla de inferencia
- sustitución de variables (del teorema o de la regla de inferencia)

En muchas ocasiones una razón descrita de manera incompleta es suficiente para explicar el paso. El sentido común es importante para saber cómo proceder en cada caso; lo importante es que el interlocutor entienda por qué el paso corresponde a una deducción correcta.

Recuérdese que, como convención, se acepta escribir = o ≡ como conectivo que precede una razón. La idea es que se están cambiando iguales por iguales, de manera que el valor de verdad de las proposiciones se mantiene equivalente. Esta licencia se permite solo en el uso de los conectivos que preceden las razones; dentro de una fórmula se sigue escribiendo ≡ para señalar la equivalencia de valores de verdad[15].

Ejemplo A: Demostraciones ecuacionales

Teo A: $p \wedge q \Rightarrow p \vee q$

Dem:

$p \wedge q \Rightarrow p \vee q$
$=$ ⟨ Def ⇒ ⟩
$\neg(p \wedge q) \vee (p \vee q)$
$=$ ⟨De Morgan, SAP⟩
$\neg p \vee \neg q \vee p \vee q$
$=$ ⟨ SAP ⟩
$p \vee \neg p \vee q \vee \neg q$
$=$ ⟨ medio excluido ⟩
$true \vee q \vee \neg q$
$=$ ⟨ true ∨ x ≡ true ⟩
$true$

□

El teorema A vale porque se muestra equivalente a `true`. Nótese cómo las justificaciones citan teoremas aceptados, aplicados a la línea anterior. En realidad, se entiende que estos teoremas son equivalencias y se aplican a una parte de la línea de la demostración con la regla de Leibniz.

Según se ha definido, las únicas dos reglas de inferencia que se pueden usar son la de sustitución y la de Leibniz[16].

Nótese que, aunque las razones son textos incompletos en la descripción de la corrección de los pasos, la demostración debería entenderse con algo de sentido común.

Teo B: $p \wedge (\neg q \Rightarrow p) \equiv p$

Dem:

$p \wedge (\neg q \Rightarrow p) \equiv p$
$=$ ⟨Def ⇒⟩
$p \wedge (\neg\neg q \vee p) \equiv p$
$=$ ⟨Doble ¬⟩
$p \wedge (q \vee p) \equiv p$
$=$ ⟨ ∨-Conmutatividad ⟩
$p \wedge (p \vee q) \equiv p$

□

El teorema B vale porque se muestra equivalente a un teorema conocido (cf. 2.4.1 Absorción).

---

[15] Una fórmula como `(1=2) ≡ false` debería aceptarse, en una extensión del lenguaje que permitiera un símbolo de igualdad = y constantes de tipo aritmético. En este caso, estaría mal escribir `(1=2) = false`.

[16] En realidad se están usando la reflexividad, asociatividad y simetría de la equivalencia, pero esto está sobreentendido en la forma de escribir, de usar y de entender el esquema de prueba.

Teo C: $\neg p \Rightarrow q \equiv p \vee q$

Dem:

$\neg p \Rightarrow q$
$=$ ⟨Def ⇒⟩
$\neg\neg p \vee q$
$=$ ⟨Doble ¬⟩
$p \vee q$

□

Este último método de prueba es útil si la fórmula que se quiere demostrar incluye una equivalencia. En este caso se empieza por un lado de la equivalencia y se muestra que el otro lado es equivalente al del comienzo. Es un buen consejo empezar por la parte más complicada de la equivalencia y buscar llegar a la más simple. Como otro ejemplo para ilustrar la técnica, el teorema B se puede probar así:

Teo B': $p \wedge (\neg q \Rightarrow p) \equiv p$

Dem:

$p \wedge (\neg q \Rightarrow p)$
$=$ ⟨Def ⇒⟩
$p \wedge (\neg\neg q \vee p)$
$=$ ⟨Doble ¬⟩
$p \wedge (q \vee p)$
$=$ ⟨ ∨-Conmutatividad ⟩
$p \wedge (p \vee q)$
$=$ ⟨Absorción⟩
p

□

Nótese cómo no se repite la equivalencia con p en todas las líneas de la demostración. La ley de absorción se usa como justificación del paso final.

§

Ejercicios 2.4.3

Demuestre los siguientes teoremas usando el cálculo ecuacional:

1 $x \wedge y \equiv x \vee y \equiv x \equiv y$

2 $(x \Rightarrow y) \wedge \neg(x \equiv y) \Rightarrow y$

3 $p \wedge (p \Rightarrow q) \wedge (q \Rightarrow r) \Rightarrow r$

4 $p \vee \neg p \equiv ((p \vee q) \wedge \neg(\neg p \wedge (\neg q \vee \neg r))) \vee (\neg p \wedge \neg q) \vee (\neg p \wedge \neg r)$

5 $(p \vee q) \wedge (\neg p \wedge (\neg p \wedge q)) \equiv (\neg p \wedge q)$

6 $p \Rightarrow (q \Rightarrow p) \equiv \neg p \Rightarrow (p \Rightarrow q)$

7 $(p \Rightarrow q) \wedge (r \Rightarrow q) \equiv (p \vee r) \Rightarrow q$

2.4.4 Otras formas de demostración

El esquema básico de demostración con equivalencias es *completo*, en el sentido de que toda fórmula válida es un teorema del cálculo ecuacional explicado en 2.4.1 y 2.4.2 (cf. [Tou2008]). Sin embargo, este importante resultado teórico no dice que las demostraciones tengan que ser fáciles de entender o cortas de escribir. De hecho, no es así en muchos casos.

Una manera de mejorar la práctica de la construcción de demostraciones es enriquecer los esquemas de prueba, permitiendo más variedad de estructuras de demostraciones, cuya justificación y corrección pueda explicarse en términos de las que se hacen de la forma básica.

A la forma básica de demostración con equivalencias se pueden agregar dos variantes importantes: el uso de lemas y el uso del teorema de la deducción. Por otro lado, se mostrarán otras variantes de esquemas, que permiten construir pruebas con técnicas de razonamiento específicas que, por lo mismo, ameritan la introducción de notación propia para la técnica dada.

2.4.4.1 Lemas

En la prueba de un teorema α se puede pensar en usar otro teorema conocido β, más la regla de Leibniz, para justificar un paso específico. Nótese que para esto se supone que uno conoce el teorema β antes de demostrar α.

Sin embargo, es concebible que β no esté en una lista de teoremas conocidos, pero que sí sea un teorema. Entonces, todo lo que hay que hacer es mostrar β como teorema, para poder utilizarlo como razón. Es decir, se prueba

Lema β
Dem:
…

Y se usa en la demostración de α en algún paso:

…
E_{i-1}
= $\langle\ \beta\ \rangle$
E_i
…

La prueba de un teorema puede tener muchos lemas que la apoyen. De hecho, la demostración de un lema puede requerir la demostración de otro. Lo que importa es que se evidencie que cada paso de deducción se apoya en teoremas que se pueden demostrar con anterioridad a su uso (cf. 2.6).

2.4.4.2 El Metateorema de la deducción

Un resultado de la teoría de la demostración -que no se probará- relaciona la deducción con la implicación de la siguiente forma:

Metateorema de la deducción

$$\alpha_1, \ldots, \alpha_k \vdash \beta \quad \text{si y solo si} \quad \vdash \alpha_1 \wedge \ldots \wedge \alpha_k \Rightarrow \beta$$

□

Primero, hay que entender la notación. Con

$$\alpha_1, \ldots, \alpha_k \vdash \beta$$

se quiere significar que la expresión β se puede demostrar formalmente a partir de los axiomas adicionando las hipótesis[17] $\alpha_1,\ldots,\alpha_k$ y usando las reglas de inferencia. Cuando k=0, la fórmula colapsa en

$$\vdash \beta$$

y esto quiere decir que β es demostrable usando solamente los axiomas y las reglas de siempre. Es decir, β es un teorema.

El metateorema de la deducción es una verdad sobre lo que significa deducir o calcular teoremas (es decir, un teorema sobre los teoremas). Es, en esencia, la misma justificación de cuándo se puede introducir al cálculo una regla de inferencia.

El uso práctico del metateorema de la deducción es de derecha a izquierda: si se desea mostrar un teorema de la forma

$$\alpha_1 \wedge \ldots \wedge \alpha_k \Rightarrow \beta$$

el metateorema garantiza que ese teorema es válido si se puede mostrar que

$$\alpha_1,\ldots,\alpha_k \vdash \beta$$

Esto último quiere decir que, si se suponen como hipótesis (o sea, como fórmulas válidas) $\alpha_1,\ldots,\alpha_k$, se debe poder deducir β. Nótese que, cuando alguna de las hipótesis $\alpha_1,\ldots,\alpha_k$ es falsa, el teorema $\alpha_1 \wedge \ldots \wedge \alpha_k \Rightarrow \beta$ es verdad sin importar el valor de verdad de β.

Normalmente, el uso de este teorema da lugar a demostraciones más cortas y más claras. Cuando se usa, el esquema en que se escribe lo resalta, señalando las hipótesis que se pueden usar (también estaría bien si se comenta lo que se pretende demostrar):

Teo: $\alpha_1 \wedge \ldots \wedge \alpha_k \Rightarrow \beta$
Dem:
Hip: $\alpha_1, \ldots, \alpha_k$ // A demostrar: β
⟨ Demostración de β, usando $\alpha_1,\ldots,\alpha_k$ ⟩

□

Ejemplo A: Ejemplo de uso del Teorema

Teo A': $p \wedge q \Rightarrow p \vee q$
Dem:
Hip: p, q // A demostrar: $p \vee q$
$p \vee q$
= ⟨Hip p⟩
$true \vee q$
= ⟨ $true \vee x \equiv true$ ⟩
$true$

□

La prueba es mucho más corta que la del Ejemplo A en 2.4.3. Nótese cómo, además, no se usaron todas las hipótesis en la prueba.

§

[17] "Adicionar hipótesis" es equivalente a "enriquecer, localmente, el conjunto de axiomas". Es decir, se trata de suponer, solo para la demostración que se hace, que las hipótesis son fórmulas que se pueden tomar como verdades.

2.4.4.3 Relajación del esquema de pruebas

Se puede probar que el operador de implicación $\Rightarrow$ cumple los siguientes teoremas, que reflejan una especie de "inter-transitividad" entre los operadores de implicación y de equivalencia:

$\vdash$ ((p $\Rightarrow$ q) $\wedge$ (q $\equiv$ r)) $\Rightarrow$ (p $\Rightarrow$ r)
$\vdash$ ((p $\equiv$ q) $\wedge$ (q $\Rightarrow$ r)) $\Rightarrow$ (p $\Rightarrow$ r)
$\vdash$ ((p $\Rightarrow$ q) $\wedge$ (q $\Rightarrow$ r)) $\Rightarrow$ (p $\Rightarrow$ r)

Usando el Teorema de la deducción, se concluye que

p $\Rightarrow$ q, q $\equiv$ r $\vdash$ p $\Rightarrow$ r
p $\equiv$ q, q $\Rightarrow$ r $\vdash$ p $\Rightarrow$ r
p $\Rightarrow$ q, q $\Rightarrow$ r $\vdash$ p $\Rightarrow$ r

La verdad de estas propiedades justifica que se relaje el esquema de pruebas para permitir implicaciones ($\Rightarrow$) para conectar razones en demostraciones de implicaciones (como p $\Rightarrow$ r).

Es decir, se pueden tener demostraciones de la forma:

Teo : α
Dem:
E_0
= ⟨razón 1⟩
…
$\Rightarrow$ ⟨razón i⟩
…
$\Rightarrow$ ⟨razón j⟩
…
= ⟨razón k⟩
E_k

□

En este caso, se espera que α tenga la forma $E_0 \Rightarrow E_k$. Obsérvese que la prueba deja de ser ecuacional, de modo que, de ninguna manera, se demuestra que $E_k \Rightarrow E_0$.

Como el operador de consecuencia $\Leftarrow$ satisface teoremas simétricos a los mencionados para la implicación, también se permiten demostraciones que usen este conectivo precediendo las razones:

Teo : α
Dem:
E_0
= ⟨razón 1⟩
…
$\Leftarrow$ ⟨razón i⟩
…
$\Leftarrow$ ⟨razón j⟩
…
= ⟨razón k⟩
E_k

□

Lo que no se permite, porque es incorrecto, es la mezcla de conectivos $\Rightarrow$ y $\Leftarrow$ en una misma demostración.

2.4.4.4 Debilitamiento / Fortalecimiento

Si $\alpha \Rightarrow \beta$ es válida, se dice que β es *más débil* que α. De igual manera, se dice que α es *más fuerte* que β. Con esto se quiere significar que, en una implicación, el *antecedente* (la parte izquierda) exige más que el *consecuente* (la parte derecha).

La proposición más débil posible es `true`. La más fuerte es `false`[18].

Una manera natural de usar razones que tengan implicaciones se da cuando se utilizan las siguientes, que corresponden a teoremas llamados de *debilitamiento* o *fortalecimiento* (según se mire) de lo afirmado:

$\vdash$ p $\Rightarrow$ p $\vee$ q
$\vdash$ p $\wedge$ q $\Rightarrow$ p
$\vdash$ p $\wedge$ q $\Rightarrow$ p $\vee$ q
$\vdash$ p $\vee$ (q $\wedge$ r) $\Rightarrow$ p $\vee$ q
$\vdash$ p $\wedge$ q $\Rightarrow$ p $\wedge$ (q $\vee$ r)

Más aun, por el Metateorema de la Deducción se puede afirmar que

p $\vdash$ p $\vee$ q
p, q $\vdash$ p
p, q $\vdash$ p $\vee$ q
p $\vee$ (q $\wedge$ r) $\vdash$ p $\vee$ q
p, q $\vdash$ p $\wedge$ (q $\vee$ r)

De esta manera, los teoremas de debilitamiento / fortalecimiento citados permiten relajar las pruebas, si se escriben pasos de la forma –por ejemplo-

…
α
$\Rightarrow$ $\langle$ Debilitamiento $\rangle$
$\alpha \vee \beta$
…

2.4.4.5 Modus Ponens

Una fuente muy natural de implicaciones en las demostraciones es que el teorema

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

justifica el uso de la regla de *Modus Ponens*:

p, p $\Rightarrow$ q $\vdash$ q

Es decir, el cálculo permite demostrar un teorema que, al ser una fórmula de una implicación que resulta ser válida, da lugar al Modus Ponens, como una nueva regla de inferencia que se puede usar en las demostraciones.

Como el Modus Ponens requiere la validez de dos hipótesis para ser aplicado, una de ellas es la de la fórmula en cuestión y otra, la de un teorema ya probado. Lo más usual es que la fórmula que se ha derivado corresponda a p y que el otro teorema que se aplique sea uno de la forma p$\Rightarrow$q.

2.4.4.6 Prueba por casos

La disyunción y la implicación cumplen el siguiente teorema:

---

[18] Una forma de entender esto es pensar que `true` no exige nada para ser cumplido, y siempre se satisface, mientras que `false` exige tanto que nada lo satisface.

$\vdash \ (p \Rightarrow r) \wedge (q \Rightarrow r) \equiv (p \vee q) \Rightarrow r$

Al interpretar el resultado:
- la parte izquierda está diciendo que r es una consecuencia tanto de p como de q. O bien, en caso de que valga p, deberá valer r. Y, en caso de que valga q, también deberá valer r
- la parte derecha está diciendo que, en caso de que valgan p o q, deberá valer r.

Una situación muy común en que esto ocurre se da cuando $q \equiv \neg p$. O sea, por la Regla de Sustitución, también es un teorema:

$$(p \Rightarrow r) \wedge (\neg p \Rightarrow r) \equiv (p \vee \neg p) \Rightarrow r$$

Y, claramente, también:

$$(p \Rightarrow r) \wedge (\neg p \Rightarrow r) \equiv r$$

Ahora, la parte izquierda está diciendo que r vale, valga o no p. Esto justifica un esquema de prueba por casos en el que, para probar r, se puede muestre que r vale si la hipótesis p es añadida, pero también vale cuando la hipótesis $\neg p$ es añadida.

Con una sencilla generalización de lo anterior se justifica el siguiente esquema de prueba:

Teo r
Dem:
Casos: $p_1, \ldots, p_m$
  ⟨Demostración de $p_1 \vee \ldots \vee p_m$⟩

Caso p_1:
  ⟨Demostración de $p_1 \Rightarrow r$ ⟩
…

Caso p_m:
  ⟨Demostración de $p_m \Rightarrow r$ ⟩

□

La demostración de $p_1 \vee \ldots \vee p_m$ comprueba el *cubrimiento* de los casos considerados, es decir, que éstos cubren todas las posibilidades (porque la disyunción de los casos es true). Cuando el cubrimiento es evidente -por ejemplo, si hay dos casos contradictorios, p, $\neg p$- se puede omitir su demostración.

2.4.4.7 Prueba por contrapositiva

El siguiente teorema conocido como la ley de la *contrapositiva* o *contrarrecíproca*, se puede demostrar:
$\vdash \ (p \Rightarrow q) \equiv (\neg q \Rightarrow \neg p)$

La validez de este resultado da lugar a un esquema de prueba de la forma:

Teo $p \Rightarrow q$
Dem:
Por contrapositiva.
Hip: $\neg q$ // A demostrar: $\neg p$
  ⟨Demostración de $\neg p$⟩

□

Este método puede ser útil cuando lo que se prueba involucra negaciones que pueden eliminarse fácilmente con el axioma de doble negación ($\neg\neg x \equiv x$).

2.4.4.8 Prueba por contradicción

Como un caso especial de contrapositiva, es un teorema

$$(\neg p \Rightarrow \text{false}) \equiv (\neg \text{false} \Rightarrow \neg\neg p)$$

A partir de esto, es fácil llegar al teorema

$$(\neg p \Rightarrow \text{false}) \equiv p$$

que justifica la introducción de un nuevo formato, para las llamadas, pruebas *por contradicción*. En este caso, para probar p se supone ¬p y se trata de mostrar que `false` vale. El esquema es como sigue:

Teo: r
Dem:
Por contradicción
Hip: ¬r
⟨Demostración de false⟩

□

Ejercicios 2.4.4

Demuestre los siguientes teoremas con el cálculo deductivo extendido con los esquemas de demostración añadidos en 2.4.4:

1 $p \wedge q \wedge r \Rightarrow p$

2 $p \wedge \neg(\neg r \vee \neg p) \Rightarrow r \wedge p$

3 $(x \vee y) \wedge (x \equiv y) \Rightarrow (x \Rightarrow y)$

4 $(p \vee q) \wedge (p \equiv q) \Rightarrow p \wedge q$

5 $p \wedge (p \Rightarrow q) \wedge (q \Rightarrow r) \Rightarrow r$

6 $(p \Rightarrow q) \wedge (r \Rightarrow s) \Rightarrow (p \vee r \Rightarrow q \vee s)$

2.5 MÁS EJEMPLOS DE DEMOSTRACIONES

2.5.1 Problema A

Considérese el problema del Ejemplo C de 2.2.1. Se trataba de mostrar la corrección de la argumentación siguiente:

Si Supermán fuera capaz y quisiera prevenir el mal, él lo prevendría.
Si Supermán fuera incapaz de prevenir el mal, sería impotente; si él no quisiera prevenir el mal, sería malévolo.
Supermán no previene el mal.
Si Supermán existe, no es impotente ni malévolo.

Entonces,

Supermán no existe.

Allí se vio que la corrección del razonamiento corresponde a mostrar que la fórmula

$$F0 \wedge F1 \wedge F2 \wedge F3 \Rightarrow F4$$

sea válida, donde (la tercera columna recuerda la interpretación de las hipótesis):

F0	$a \wedge w \Rightarrow p$	Si Supermán fuera capaz y quisiera prevenir el mal, él lo prevendría
F1	$(\neg a \Rightarrow i) \wedge (\neg w \Rightarrow m)$	Si Supermán fuera incapaz de prevenir el mal, sería impotente y si Supermán no quisiera prevenir el mal, sería malévolo
F2	$\neg p$	Supermán no previene el mal
F3	$e \Rightarrow \neg i \wedge \neg m$	Si Supermán existe, no es impotente ni malévolo
F4	$\neg e$	Superman no existe

La pregunta equivale a decidir si es correcto que:

$$F0 \wedge F1 \wedge F2 \wedge F3 \Rightarrow F4.$$

o, de forma equivalente, buscando una prueba con hipótesis:

$$F0, F1, F2, F3 \vdash F4.$$

Nótese que la hipótesis F1 se puede desglosar en 2 hipótesis equivalentes
(F1a) $\neg a \Rightarrow i$
(F1b) $\neg w \Rightarrow m$

La demostración se puede esbozar planeando efectuar los siguientes pasos:
1 Observar que $\neg p$ vale, por F2.
2 Con F0, por contrapositiva, mostrar, $\neg(a \wedge w)$.
3 Por DeMorgan, concluir que valen $\neg a$ o $\neg w$
4 Con F1a mostrar i; con F1b, mostrar m.
5 Afirmar que $i \vee m$ vale.
6 Con F3 y contrapositiva concluir $\neg e$.

Lema 0: $\neg p \Rightarrow \neg a \vee \neg w$
Dem:
Hip: F0, F1a, F1b, F2, F3
$\quad$ true
= $\quad$ ⟨Hip F0: $a \wedge w \Rightarrow p$⟩
$\quad$ $a \wedge w \Rightarrow p$
= $\quad$ ⟨Contrapositiva⟩
$\quad$ $\neg p \Rightarrow \neg(a \wedge w)$
= $\quad$ ⟨DeMorgan⟩
$\quad$ $\neg p \Rightarrow \neg a \vee \neg w$

□

Lema 1: $\neg a \vee \neg w$
Dem:
Hip: F0, F1a, F1b, F2, F3, Lema 0
$\quad$ true
= $\quad$ ⟨Hip F2: $\neg p$⟩
$\quad$ $\neg p$
⇒ $\quad$ ⟨Lema 0: $\neg p \Rightarrow \neg a \vee \neg w$; Modus Ponens⟩
$\quad$ $\neg a \vee \neg w$

□

Lema 2: $i \vee m$
Dem:
Hip: F0, F1a, F1b, F2, F3, Lema_0, Lema_1
Casos: $\neg a$, $\neg w$

```
      ¬a ∨ ¬w
=           ⟨Hip: Lema_1: ¬a ∨ ¬w⟩
      true

   Caso: ¬a
      true
=           ⟨Hip F1a: ¬a ⇒ i⟩
      ¬a ⇒ i
⇒           ⟨Caso ¬a; Modus Ponens⟩
      i
⇒           ⟨Debilitamiento⟩
      i ∨ m

   Caso: ¬w
      true
⇒           ⟨Hip F1b: ¬w ⇒ m⟩
      ¬w ⇒ m
⇒           ⟨Caso ¬w; Modus Ponens⟩
      m
⇒           ⟨Debilitamiento⟩
      i ∨ m
```

□

Lema 3: ¬e

Dem:

```
Hip: F0, F1a, F1b, F2, F3, Lema_0, Lema_1, Lema_3
   true
=     ⟨Hip F3: e ⇒ ¬i ∧ ¬m⟩
   e ⇒ ¬i ∧ ¬m
=     ⟨Contrapositiva⟩
   ¬(¬i ∧ ¬m) ⇒ ¬e
=     ⟨ De Morgan, Doble negación, 2 veces⟩
   i ∨ m ⇒ ¬e
⇒     ⟨Hip: Lema_3: i∨m; Modus Ponens⟩
   ¬e
```

□

2.5.2 Problema B

Decidir si el siguiente razonamiento es correcto o no:

Si el banco no le da el préstamo a Pérez, éste tendrá que obtener crédito en otra parte. Pero si debe buscar crédito en otra parte, Pérez tendrá que vender el negocio. Y si vende el negocio, tendrá que irse a sembrar papas o a pedir limosna en un semáforo. Pérez nunca pedirá limosna en un semáforo. Por lo tanto, si el banco no le presta, Pérez tendrá que irse a sembrar papas.

Modelado

Definición de símbolos

p : El banco le da el préstamo a Pérez
o : Pérez tendrá que obtener crédito en otra parte
v : Pérez vende el negocio

s : Pérez se va a sembrar papas
f : Pérez pedirá limosna en un semáforo

Proposiciones

h1	$\neg p \Rightarrow o$	Si el banco no le da el préstamo a Pérez, éste tendrá que obtener crédito en otra parte
h2	$o \Rightarrow v$	Pero si debe buscar crédito en otra parte, Pérez tendrá que vender el negocio.
h3	$v \Rightarrow s \vee f$	Y si vende el negocio, tendrá que irse a sembrar papas o a pedir limosna en un semáforo.
h4	$\neg f$	Pérez nunca pedirá limosna en un semáforo
x	$\neg p \Rightarrow s$	si el banco no le presta, Pérez tendrá que irse a sembrar papas.

Se trata de decidir si:

$$h1 \wedge h2 \wedge h3 \wedge h4 \Rightarrow x$$

Una tabla de verdad usará 32 filas y, al menos, 5 columnas.

Deducción

La demostración puede seguir el siguiente plan (a demostrar $\neg p \Rightarrow s$)
1 Suponer $\neg p$. Buscar demostrar s (prueba con hipótesis).
2 Con h1, mostrar o.
3 Con h2, mostrar v.
4 Con h3, mostrar $s \vee f$.
5 Con h4, mostrar s.

Teo: $h1 \wedge h2 \wedge h3 \wedge h4 \Rightarrow x$
Hip: h1, h2, h3, h4 // A demostrar: $\neg p \Rightarrow s$
  Hip: $\neg p$ // A demostrar: s
    true
  = ⟨Hip: $\neg p$⟩
    $\neg p$
  $\Rightarrow$ ⟨Hip h1: $\neg p \Rightarrow o$; Modus Ponens⟩
    o
  $\Rightarrow$ ⟨Hip h2: $o \Rightarrow v$; Modus Ponens⟩
    v
  $\Rightarrow$ ⟨Hip h3: $v \Rightarrow s \vee f$; Modus Ponens⟩
    $s \vee f$
  = ⟨Hip h4: $\neg f$, i.e.: $f \equiv$ false⟩
    $s \vee$ false
  = ⟨$\vee$-Identidad⟩
    s

□

En el uso de la hipótesis h4, se puede remplazar f por false si se sabe que $\neg f$ es true.

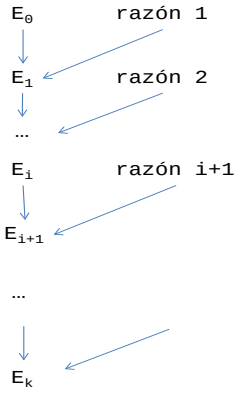
□

Ejercicios 2.5

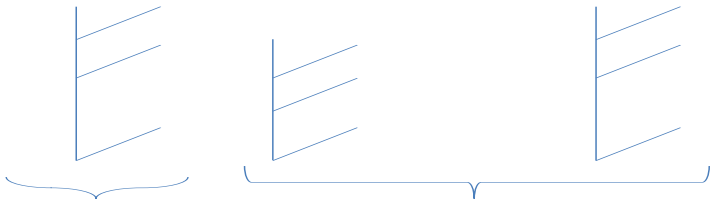
En los Ejercicios 2.2.1, desarrolle demostraciones para los problemas 2 y 3.

2.6 MÁS SOBRE DEMOSTRACIONES

Los esquemas de demostraciones que se han definido como correctos tienen como peculiaridad interesante el que siempre usan lo último deducido como base para la siguiente deducción. Estas son las llamadas *prueba lineales*, pues podrían graficarse de manera esquemática con una especie de "espina de pescado" en la que la columna vertebral es la línea de la demostración y las espinas son las razones de la prueba:



No siempre se puede pensar en demostrar de este modo. En general, se permite probar lemas que ayuden en la demostración como razones (teoremas ya mostrados) que pueden ser muy locales a la prueba particular que se hace. Los lemas se demuestran con el mismo mecanismo, de manera que, viendo todo en perspectiva, una demostración es un conjunto de "espinas de pescado" como la mostrada:



No hay métodos conocidos que garanticen que un resultado se pueda demostrar. La pequeña ventaja de esta forma de pruebas es que, al saber que siempre se debe buscar hacer algo con lo recién demostrado, se puede tener un poco más de claridad para juzgar si se va por un camino exitoso.

Como ya se anotó, el cálculo ecuacional que responde al aparato deductivo que se ha dado aquí es *completo* para la lógica proposicional. Con los esquemas adicionales, y con la posibilidad de usar el Meatateorema de la Deducción, muchas pruebas resultan más simples que las que usan solo cálculo ecuacional puro.

La completitud del cálculo permite decir que, si una fórmula es una tautología, existe una demostración en este cálculo para esa fórmula. Sin embargo, la existencia de la demostración no tiene que conocerse de manera explícita, lo que resulta en que puede no ser fácil encontrarla De hecho, no hay método práctico conocido para encontrar estas demostraciones, para un teorema arbitrario dado. Por esta razón, el demostrar requiere usar unas destrezas que se aprenden con ejercicios, ensayos y errores.

2.6.1 Uso correcto de razones

Se ha dicho que se pueden usar como razones:

- Axiomas
- Teoremas ya demostrados
- Modus Ponens

- Debilitamiento / Fortalecimiento

Dentro de los teoremas ya demostrados hay unos que se han usado para justificar nuevos esquemas de pruebas (uso de conectivos $\Rightarrow$ precediendo razones, contrapositiva, casos, debilitamiento / fortalecimiento). Si se quiere justificar rápidamente estos esquemas, basta con hacer una tabla de verdad para el teorema correspondiente.

Un error común que ocurre en demostraciones incorrectas es el usar como razones algún teorema que, al momento de la deducción correspondiente, no se tenga por demostrado. Si ese teorema fuera un lema para probar, debería contarse con el lema antes de usarlo. Se puede aceptar que la prueba del lema sea posterior, en gracia de hacer una exposición más clara y cuidando que no se caiga, al probar el lema, en problemas similares.

A modo de referencia para este cálculo, el Apéndice B incluye una tabla de teoremas importantes que, al estudiarse, deberían ser demostrados en el orden en que se listan. En otras palabras, estos teoremas se listan con unos identificadores que los ordenan de modo que, en la demostración de un teorema no pueden aparecer como razones identificadores de teoremas cuyo identificador es posterior, en el orden listado, al identificador del teorema en cuestión.