LMD
Práctica 4. Lógica de proposiciones

Una vez iniciada la sesión de Maxima, desplegamos el menú [Archivo]>[Cargar paquete]. Entonces se abre una ventana para que seleccionemos un archivo del tipo Paquete Maxima (*.mac). Nosotros seleccionamos el tipo de archivo Paquete Lisp (*.lisp) y cargamos el paquete logic.lisp que acompaña a la práctica. Este conjunto de funciones, que ya hemos utilizado en el Tema 1, nos permitirá también trabajar con proposiciones lógicas. La versión de Maxima que estamos usando es algo antigua. En las versiones más modernas, las funciones del paquete *logic.lisp* ya vienen incorporadas en Maxima y se cargan escribiendo `load("logic.mac")`$.

Los operadores reconocidos son los siguientes:

operador	tipo	precedencia	descripción
not	prefijo	70	negación (NEG, $\neg$)
and	infijo	65	conjunción (AND, $\wedge$)
or	infijo	60	diyunción (OR, $\vee$)
implies	infijo	59	implicación ($\rightarrow$)
eq	infijo	58	equivalencia ($\leftrightarrow$)

La primera columna nos indica el nombre del operador lógico en el paquete logic.lisp y la segunda la forma cómo éste se escribe. Así, si se trata de un operador prefijo significa que dicho operador va delante de los operandos, mientras que si es infijo se escribe entre los dos operandos a los que se aplica.

La tercera columna nos dice la precedencia ó prioridad del operador. Los operadores con mayor precedencia se evalúan antes y en caso de empate se consideran de izquierda a derecha.

La última columna indica otros símbolos o notaciones con los que también se suele representar a los operadores descritos en textos de lógica.

A lo largo de esta práctica usaremos las letras `a`, `b`, `c` ... para denotar proposiciones lógicas.

Algunos ejemplos de proposiciones lógicas escritas con la sintaxis que acabamos de definir, y que en este momento no ha de introducir en Maxima, son los siguientes:

```
not a,  a or b,  a implies (b and not c),  (a eq b) eq c.
```

El valor de verdad que puede tomar una proposición en el paquete logic.lisp se representa mediante true, si ésta es verdadera, y false, si es falsa.

Además las palabras true y false serán usadas para denotar una proposición tautológica y una proposición contradictoria o contradicción, respectivamente.

Este doble uso de las palabras true y false no supone ningún problema, pues siempre

que nos encontremos con alguna de ellas, el contexto nos dirá si se trata de una proposición o bien de un valor de verdad.

Comenzamos estudiando el comando characteristic_vector(prop) que nos proporciona una lista de valores lógicos `true` o `false` que corresponde a la columna que aparece más a la derecha en la tabla de verdad de la proposición prop. Esta lista se construye considerando las variables que aparecen en la proposición ordenadas alfabéticamente de izquierda a derecha. Pruebe el siguiente ejemplo:

```
(%ixx) characteristic_vector(a implies b);
```

Vea cómo se obtiene la lista `[true,true,false,true]`, donde `false` aparece en tercera posición, pues las variables `a` y `b` (en este orden) se evalúan con las combinaciones $00, 01, 10, 11$.

Pruebe ahora con:

```
(%ixx) characteristic_vector(b implies a);
```

Ahora resulta la lista `[true,false,true,true]`, donde `false` va en segunda posición, ya que las variables `a` y `b` (en éste orden) se siguen evaluando con las combinaciones $00, 01, 10, 11$.

El siguiente ejemplo nos dice que toda proposición de la forma $\alpha \to (\beta \to \alpha)$ es una tautología.

```
(%ixx) characteristic_vector(a implies (b implies a));
```

Como sabemos, cada uno de los operadores lógicos `or` y `and` verifica la propiedad asociativa y por tanto no requieren el uso de paréntesis.

```
(%ixx) characteristic_vector(a or b or c);
```

```
(%ixx) characteristic_vector(a and b and c);
```

Introduzca los siguientes ejemplos.

```
(%ixx) characteristic_vector(a or b and c);
```

```
(%ixx) characteristic_vector(a or (b and c));
```

Como puede observar, resulta la misma tabla de verdad, ya que según la descripción inicial de los operadores lógicos, el `and` (con un valor de 65) tiene más prioridad que el `or` (con un valor de 60).

Introduzca el siguiente ejemplo.

```
(%ixx) characteristic_vector((a or b) and c);
```

Ahora vemos que la tabla es distinta de las dos anteriores. Con los paréntesis estamos forzando a que el operador `or` sea evaluado antes que el `and`.

```
(%ixx) characteristic_vector(a implies (b implies c));
```

```
(%ixx) characteristic_vector((a implies b) implies c);
```

Por tanto nos damos cuenta de que el operador `implies` ($\to$) no verifica la propiedad asociativa.

```
(%ixx) characteristic_vector(a implies b implies c);
```

Si comparamos este ejemplo con los dos anteriores, vemos que escribir la expresión `a implies b implies c` es equivalente a escribir `(a implies b) implies c`. Tal y como decíamos antes, en caso de empate entre operadores, se asigna más prioridad de izquierda a derecha.

Hemos visto que el comando characteristic_vector calcula una lista de valores lógicos que representan (la columna más a la derecha en) la tabla de verdad de una proposición lógica a partir de las variables proposicionales con las que se construye dicha proposición. De modo más general, este comando admite el siguiente formato:

characteristic_vector(prop, v_1,...,v_n)

Éste nos proporciona la "tabla de verdad" de la proposición prop respecto de las variables v_1,...,v_n que le indiquemos (ordenadas de esta forma). Por ejemplo, es indiferente escribir cualquiera de los dos comandos siguientes:

```
(%ixx) characteristic_vector(a implies b);
```

```
(%ixx) characteristic_vector(a implies b, a,b);
```

Comprobamos ahora que el orden de las variables indicadas también influye a la hora de construir la tabla.

```
(%ixx) characteristic_vector(a implies b, b,a);
```

Por supuesto, podemos añadir otras variables las cuales no tienen por qué aparecer en la fórmula dada.

```
(%ixx) characteristic_vector(a implies b, a,b,c);
```

```
(%ixx) characteristic_vector(a implies b, a);
```

```
(%ixx) characteristic_vector(a implies b, b);
```

En los dos últimos ejemplos hemos construído tablas de verdad de una fórmula respecto de una sóla de sus variables. De ahí que los valores resultantes vengan dados en función de otras variables que aparecen en tales fórmulas.

Ahora ya entenderá la salidas devueltas por los comandos siguientes.

```
(%ixx) characteristic_vector(a implies b, c);
```

```
(%ixx) characteristic_vector(a implies b, c,c);
```

```
(%ixx) characteristic_vector(true, a);
```

```
(%ixx) characteristic_vector(true, a,b);
```

Dos proposiciones lógicas α y β pertenecientes a un lenguaje de proposiciones $\mathcal{F}(X)$, se dice que son equivalentes si para cualquier interpretación I sobre $\mathcal{F}(X)$ se verifica que $I(\alpha) = I(\beta)$. Dicho de otro modo, α y β son equivalentes si la proposición $\alpha \leftrightarrow \beta$ es una tautología.

El comando logic_equiv($prop_1$, $prop_2$) devuelve true si las proposiciones representadas por $prop_1$ y $prop_2$ son equivalentes, y false en caso contrario.

Como consencuencia de lo anterior, una proposición representada por prop es una tautología si y sólo si logic_equiv(prop, true) devuelve true. Observe cómo aquí estamos usando la palabra true para dos cosas diferentes, aunque el contexto nos dice lo que es en cada caso.

De manera similar, prop es una contradicción si y sólo si logic_equiv(prop, false) devuelve true.

Introduzca las siguientes instrucciones.

```
(%ixx) logic_equiv(a implies b, not a or b);
(%ixx) logic_equiv(a implies (b implies a), true);
(%ixx) logic_equiv(a and not a, false);
(%ixx) logic_equiv(a eq not a, false);
```

Con el segundo comando hemos comprobado de una manera más simple algo que ya hicimos anteriormente, y es que toda fórmula del tipo $\alpha \to (\beta \to \alpha)$ es una tautología.

Con el tercero y cuarto, hemos visto que todas las fórmulas del tipo $\alpha \wedge \neg\alpha$ así como $\alpha \leftrightarrow \neg\alpha$ son contradicciones.

Como en ninguno de los dos comandos siguientes se obtiene el valor `true`, deducimos que la proposición $\alpha \vee \beta$ es contingente.

```
(%ixx) logic_equiv(a or b, true);
(%ixx) logic_equiv(a or b, false);
```

Ejercicio. Clasifique cada una de las proposiciones siguientes:

1. $\neg(P \wedge \neg Q \wedge R) \to (R \to (P \to Q))$
2. $(P \vee Q) \to (P \wedge Q)$
3. $(Q \vee \neg R) \to (P \to P)$
4. $(P \wedge Q) \wedge \neg(\neg P \to Q)$
5. $P \leftrightarrow (Q \leftrightarrow (P \leftrightarrow Q))$
6. $(P \to (Q \vee R)) \vee (P \to (Q \wedge R))$
7. $(P \to Q) \to ((Q \to R) \to (P \to R))$
8. $\neg((P \to Q) \vee (P \to (\neg Q)))$ □

A partir de las definiciones es fácil observar que un conjunto de proposiciones lógicas $\Omega = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ es satisfacible, si y sólo si, la proposición lógica $\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$ es satisfacible. También que Ω es insatisfacible, si y sólo si, la proposición $\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$ es una contradicción.

Ejercicio. Determine cuáles de los conjuntos siguientes de proposiciones son satisfacibles:

1. $\Big\{\neg(P \wedge \neg Q \wedge R) \to (R \to (P \to Q)),\ Q \vee \neg R) \to (S \to S)\Big\}$.

2. $\Big\{P \vee Q \to P \wedge Q \wedge R,\ Q \to P \wedge \neg R \wedge \neg S\Big\}$.

3. $\Big\{R \leftrightarrow R,\ P \wedge Q \wedge \neg(\neg P \to Q),\ Q \vee \neg S \to P\Big\}$.

4. $\Big\{P \leftrightarrow (Q \leftrightarrow (P \leftrightarrow Q)),\ ((P \leftrightarrow Q) \leftrightarrow P) \leftrightarrow Q\Big\}$.

5. $\Big\{(P \to (\neg Q \to R)) \vee (P \to (\neg R \to Q)),$
   $(P \to Q) \to ((Q \to R) \to (P \to R))\Big\}$. □

Dadas las proposiciones lógicas $\alpha_1, \alpha_2, \ldots, \alpha_n, \beta$, sabemos que el conjunto

$$\Omega = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$$

implica semánticamente a β, si y sólo si, el conjunto ampliado $\{\alpha_1, \alpha_2, \ldots, \alpha_n, \neg\beta\}$ es insatisfacible. También se dice que β es consecuencia lógica de Ω y se denota por $\Omega \models \beta$. Como sabemos de teoría, ésto a su vez equivale a que la proposición

$$\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n \wedge \neg\beta$$

sea una contradicción.

Ejercicio. Dado el conjunto de proposiciones

$$\Omega = \Big\{P \to (Q \wedge R),\ Q \to S,\ P \wedge T\Big\},$$

aplique la idea del párrafo anterior para determinar cuáles de las proposiciones β siguientes son consecuencia lógica de Ω.

1. $\beta = S$.
2. $\beta = \neg T \wedge R$.
3. $\beta = \neg S \to (\neg P \vee T)$.
4. $\beta = \neg T \vee Q$.
5. $\beta = S \leftrightarrow Q$.
6. $\beta = \neg Q \wedge S$.
7. $\beta = S \to Q$.
8. $\beta = S \vee \neg R$. □

A continuación vamos a ver comandos que transforman proposiciones en otras equivalentes.

El comando boolean_form(prop) escribe la proposición dada por prop sólo en función de los operadores not, or ó and. La única excepción se produce cuando prop es true ó false. Pruebe las siguientes instrucciones.

```
(%ixx) boolean_form(a implies b);
(%ixx) boolean_form(a eq b);
(%ixx) boolean_form((a implies b) implies c);
(%ixx) boolean_form(true);
(%ixx) boolean_form(((a implies b) implies c) implies d);
```

Como se puede observar en las respuestas obtenidas, no se intenta simplificar más nada.

El comando demorgan(prop) aplica las leyes de De Morgan a la proposición prop.

```
(%ixx) demorgan(not (a or b));
(%ixx) demorgan(not (a and b));
(%ixx) demorgan(not (a and b and c));
(%ixx) demorgan(not not a);
```

Éste comando hay que usarlo tras haber aplicado boolean_form, pues de lo contrario no transforma la expresión.

```
(%ixx) demorgan(a implies b);
(%ixx) demorgan(a eq b);
(%ixx) demorgan(boolean_form((a implies b) implies c));
```

Aunque la respuesta puede que no sea todo lo satisfactoria que uno espera.

```
(%ixx) demorgan(boolean_form(((a implies b) implies c) implies d));
```

Si tenemos el conjunto de variables proposicionales $X = \{a, b, c\}$ y la interpretación I sobre $\mathcal{F}(X)$ tal que $\mathrm{I}(a) = 1$, $\mathrm{I}(b) = 1$, $\mathrm{I}(c) = 0$, con el siguiente comando podemos interpretar mediante I cualquier proposición lógica perteneciente a $\mathcal{F}(X)$.

```
(%ixx) I(prop):=sublis([a=true, b=true, c=false], boolean_form(prop));
```

Lo probamos con algunos ejemplos.

```
(%ixx) I(a implies b);
(%ixx) I(a implies c);
(%ixx) I(a eq b);
(%ixx) I(a eq c);
(%ixx) I((a implies c) implies c);
```

Vimos en el Tema 1 que toda función booleana distinta de la función constante **0** se escribe de manera única como suma de mintérminos, y toda función booleana distinta de la función constante **1** se escribe de manera única como producto de maxtérminos. Las expresiones resultantes son, respectivamente, la forma normal disyuntiva y la forma normal conjuntiva de la función dada.

Imitando lo que ocurre con las funciones booleanas, toda proposición lógica α se puede expresar usando los operadores lógicos $\neg, \wedge, \vee$. De este modo obtenemos la forma normal conjuntiva y la forma normal disyuntiva de α respecto de las variables proposicionales que aparecen en α.

Dichas formas normales se obtienen con los comandos pcnf y pdnf, respectivamente. Pruebe las siguientes instrucciones. Recuerde que el operador `and` tiene más prioridad que el operador `or`, por lo cual en la salida devuelta por el comando `pcnf` (en general) aparecerán paréntesis, mientras que en la salida devuelta por el comando `pdbf` no.

```
(%ixx) pcnf(a and b);
       pdnf(a and b);
       pcnf(a or b);
       pdnf(a or b);

(%ixx) pcnf(a and b and not c);
       pdnf(a and b and not c);
       pcnf(a or b or not c);
       pdnf(a or b or not c);

(%ixx) pcnf(a implies b);
       pdnf(a implies b);

(%ixx) pcnf(a eq b);
       pdnf(a eq b);

(%ixx) pcnf(((a implies b) implies c) implies d);
       pdnf(((a implies b) implies c) implies d);
```

En particular, vemos que el comando pcnf(prop) nos proporciona una forma clausulada para la proposición prop. Está claro que dicha forma clausulada será en general bastante redundante, pues cada una de sus cláusulas incluye todas las variables proposicionales que aparecen en la proposición prop. Por ejemplo, hemos visto en teoría que una forma clausulada de la proposición $\alpha = (\neg P \wedge Q \wedge R) \vee (\neg P \wedge \neg Q)$ es $\neg P \wedge (R \vee \neg Q)$. Compruebe lo dicho más arriba introduciendo el comando siguiente:

```
(%ixx) pcnf((not P and Q and R) or (not P and not Q));
```

El problema de obtener un forma clausulada lo más simple posible para una proposición lógica, es equivalente al problema (estudiado en el Tema 1) de minimizar funciones booleanas como producto de sumas.

En nuestro ejemplo, para la proposición lógica $\alpha = (\neg P \wedge Q \wedge R) \vee (\neg P \wedge \neg Q)$, tenemos la función booleana $f(x, y, z) = \overline{x}yz + \overline{x} \cdot \overline{y}$. Consideramos su función complementaria $\overline{f(x, y, z)}$ a la que aplicamos cualquiera de las técnicas de minimización como suma de productos. Por último complementamos la expresión minimal resultante y aplicamos las Leyes de De Morgan. Así, obtenemos una expresión minimal como producto de sumas para $f(x, y, z)$. De ésta, finalmente obtenemos una forma clausulada minimal para α.

Ejercicio. Obtenga una forma clausulada lo más simple posible para la proposición lógica

$$\alpha = \Big(\neg T \to (\neg P \wedge Q)\Big) \leftrightarrow \Big(\big(P \vee \neg S\big) \to \big(\neg Q \wedge R \wedge T\big)\Big).$$

□

Ejercicio. En cada uno de los apartados siguientes, encuentre una proposición lógica equivalente a la proposición dada en ese apartado y que se escriba usando sólo operadores lógicos del conjunto $\{\neg, \rightarrow\}$.

1. $P \wedge Q$
2. $P \wedge Q \wedge R$
3. $P \vee Q$
4. $P \vee Q \vee R$
5. $P \leftrightarrow Q$

□

Ejercicio. ¿Cada proposición lógica es equivalente a una única proposición lógica que se construye usando sólo operadores lógicos del conjunto $\{\neg, \rightarrow\}$? □

Ejercicio. Si α es una proposición lógica que es una contradicción, justifique que cualquier proposición lógica β es equivalente a otra en la que sólo puede aparecer el operador lógico $\rightarrow$ y/ó la proposición α. □

Ejercicio. Clasifique cada una de las proposiciones lógicas siguientes:

1. $(p \vee (p \wedge (q \vee s \vee t))) \wedge \neg p \rightarrow \neg q$.
2. $((p \vee (p \wedge (q \vee s))) \wedge \neg p) \wedge q \wedge t$.

□

Ejercicio. Dadas las proposiciones lógicas

$$\begin{aligned} \alpha_1 &= (P \wedge Q) \rightarrow (P \vee Q), \\ \alpha_2 &= P \rightarrow (\neg Q \vee R), \\ \alpha_3 &= R \rightarrow ((P \rightarrow Q) \rightarrow \neg R), \end{aligned}$$

calcule todos los pares ordenados $(i, j) \in \{1, 2, 3\} \times \{1, 2, 3\}$ tales que la proposición $\alpha_i \rightarrow \alpha_j$ no sea una tautología. □

Ejercicio. Sea el conjunto de proposiciones

$$\begin{aligned} \Omega = \ & \{\neg P \vee \neg Q \vee R, \\ & (P \rightarrow Q) \wedge R, \\ & P \rightarrow (Q \rightarrow R), \\ & (P \wedge Q \wedge R) \vee (\neg P \wedge Q \wedge R) \vee (\neg P \wedge \neg Q \wedge R), \\ & P \rightarrow P, \\ & R \rightarrow ((P \wedge Q) \rightarrow R), \\ & (Q \wedge Q) \rightarrow (P \rightarrow R), \\ & P \wedge (P \rightarrow \neg P), \\ & \neg R \rightarrow (P \rightarrow \neg Q)\}. \end{aligned}$$

Calcule el cardinal mas grande que puede tener un subconjunto Γ de Ω verificando que dos proposiciones distintas cualesquiera pertenecientes a Γ no sean lógicamente equivalentes. □

Ejercicio. Para el conjunto de proposiciones lógicas

$$\Gamma = \Big\{(P \to R) \to Q,\ (Q \vee S) \to T,\ T \to (S \to T)\Big\},$$

encuentre cuáles de las proposiciones siguientes son consecuencia lógica de Γ:

1. $R \to T$
2. $\neg T \to P$
3. $(S \vee R) \to (P \wedge T)$
4. $\neg T \wedge R$

□

Ejercicio. Dados los conjuntos de proposiciones lógicas Ω_1 y Ω_2, se dice que Ω_1 implica semánticamente a Ω_2, ó que Ω_2 es consecuencia lógica de Ω_1, si toda interpretación que satisface a Ω_1, también satisface a Ω_2. En tal caso se escribe $\Omega_1 \models \Omega_2$.

Para el conjunto $\Omega = \Big\{\neg P \to (Q \wedge \neg R),\ (R \vee T) \leftrightarrow (P \wedge \neg Q),\ (\neg T \vee S) \to R\Big\}$, estudie cuáles de los conjuntos siguientes son consecuencia lógica de Ω:

1. $\Big\{(R \vee T) \to (P \leftrightarrow \neg S),\ Q \wedge \neg R\Big\}$.
2. $\Big\{P \wedge \neg R \wedge \neg S,\ \neg(P \to (\neg Q \vee \neg R \vee \neg S \vee T))\Big\}$.
3. $\Big\{P \wedge Q \wedge S \wedge T,\ \neg Q \wedge \neg R \wedge \neg S \wedge \neg T,\ P \to \neg Q\Big\}$.
4. $\Big\{Q \wedge R \wedge \neg T \to (P \vee S),\ \neg S \to (\neg P \to (\neg Q \to (R \vee T))),$
   $(P \to \neg S) \to ((R \wedge \neg T) \to (S \to \neg P))\Big\}$.

□

Ejercicio. Resuelva los Ejercicios 6, 8, 17, 18, 25, 27, 28, 30, 31(b) del Tema 4 con ayuda de Maxima. □