PRACTICA 2. PROGRAMACIÓN EN **LISP**

Fecha de publicación: 08 de diciembre de 2011.
Versión: 2011/12/08

Forma de entrega: **Se realizará una entrega única al final del cuatrimestre por correo electrónico según las normas que se publicarán en la página web**: http://arantxa.ii.uam.es/~fdiez/docencia/11-12/IAextincion.html

- El código debe estar en un único fichero.
- La evaluación del código en el fichero no debe dar errores (recuerda, al hacer CTRL+A CTRL+E debe evaluar todo el código sin errores).
- Ninguna de las funciones debe ser destructiva.
- En LISP existen funciones para tratar listas como conjuntos: ***member, member-if, subsetp, adjoin, union, intersection, set-difference, remove-duplicates*.**
- Utilice **recursión** y ***mapcar, mapcan***.
- En el caso de que las **listas sean consideradas como conjuntos**, el **orden** en el que aparecen los elementos del conjunto **no es importante** (ej. el orden de los literales en una cláusula, el orden de las cláusulas en una FNC).
- **No** se puede utilizar *setf*.
- **No** se puede utilizar bucles explícitos (*loop*, *dolist*, etc.)
- Se puede utilizar **let** para evitar repetir código.

Para representar las proposiciones atómicas se utilizarán símbolos LISP: 'p 'q 'r 'j
No se permite el uso de 't 'NIL o de los conectores para representar proposiciones atómicas
Para representar los conectores utilizará los átomos LISP incluidos en la siguiente tabla:

Conector	**Representación**	
Disyunción	v	La letra v la primera letra de “velero”.
Conjunción	^	El *acento circunflejo*.
Implicación	=>	El símbolo *igual* seguido del *mayor estricto que*.
Bicondicional	<=>	El símbolo del *menor estricto que* seguido del símbolo *igual* seguido del *mayor estricto que*.
Negación	¬	El símbolo de la *negación* (se obtiene con un teclado estándar pulsando simultáneamente Alt Gr y la tecla con el 6)

- Se utilizará bien notación infijo o notación prefijo (como en las expresiones LISP) para construir fórmulas bien formadas no atómicas.
- No se utilizarán átomos con valor de verdad constante en las fórmulas (True, False)
- No se utilizarán reglas implícitas de precedencia para la aplicación de conectores. Se utilizarán paréntesis para marcar el orden de aplicación de los conectores en las FBFs.
- Uso de paréntesis:
    - Son aceptables FBFs que asuman la asociatividad de v y ^

1. ((a ^ b) ^ c ^ d)
2. (a v ((¬ b) v c v d) v (a ^ b ^ (¬ q)))

- No son aceptables fórmulas con paréntesis redundantes:
  1. (a)
  2. ((a))

- Argumentos para los conectores
  - La negación es un operador unario (toma un solo argumento).
  - La implicación y bicondicional son operadores binarios (toman dos argumento).
  - La disyunción y conjunción son operadores n-arios.
- A continuación se le muestran algunos ejemplos en notación prefijo:
  - (¬ p)
  - (=> (¬ p) q)
  - (v (=> (¬ p) q) (¬ r) (<=> a b))
  - (^ p (¬ t) (v p q r s))
- Por convención, se aceptarán como FBFs en formato prefijo (pero no en formato infijo) aquellas en las que la conjunción o la disyunción estén vacías
  - (v) ; su valor de verdad es False
  - (^) ; su valor de verdad es True

  o tengan un único argumento.

  Sea w una fórmula bien formada
  - (v w)
  - (^ w)

  Son FBF distintas que tienen el mismo valor de verdad que w

En lógica proposicional se define:

- **Literal**
  - Literal positivo: Un átomo (LISP: un átomo distinto a los que representan conectores. Ej. 'p)
  - Literal negativo: negación de un literal positivo (LISP: lista cuyo 1er elemento es el conector que representa la negación y cuyo segundo elemento es un literal positivo Ej. '(¬ p)).
- **Cláusula** (cláusula disyuntiva): disyunción de literales.
- **Forma normal conjuntiva (FNC):** Fórmula bien formada que consiste en una conjunción de cláusulas disyuntivas.

Al comienzo del programa se deben definir los conectores y los predicados para evaluar si una expresión LISP es un conector, si se trata de un conector que toma 1 argumento (unario), 2 argumentos (binario) o n argumentos (n ≥ 0, enario).

```
(defconstant +bicond+ '<=>)
(defconstant +cond+   '=>)
(defconstant +and+    '^)
(defconstant +or+     'v)
(defconstant +not+    '¬)

(defun conector-p (x)
  (or (conector-unario-p  x)
      (conector-binario-p x)
      (conector-enario-p  x)))

(defun conector-unario-p (x)
```

```
  (eql x +not+))

(defun conector-binario-p (x)
  (or (eql x +bicond+) (eql x +cond+)))

(defun conector-enario-p (x)
  (or (eql x +and+) (eql x +or+)))
```

1. Escriba código LISP para determinar si una FBF es un literal positivo.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; EJERCICIO 1
;; Predicado para determinar si una FBF es un literal positivo
;;
;; RECIBE   : FBF en formato prefijo
;; EVALÚA A : T si FBF es un literal positivo, NIL en caso
;;            contrario.

(defun literal-positivo-p (fbf) ...)

(literal-positivo-p 'p)
evalúa a T
(literal-positivo-p 'T)
(literal-positivo-p 'NIL)
(literal-positivo-p '¬)
(literal-positivo-p '(p))
(literal-positivo-p '(¬ p))
(literal-positivo-p '(¬ (v p q)))
evalúan a NIL
```

2. Escriba código LISP para determinar si una FBF es un literal negativo.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; EJERCICIO 2
;; Predicado para determinar si una FBF es un literal negativo
;;
;; RECIBE   : FBF en formato prefijo
;; EVALÚA A : T si FBF es un literal negativo, NIL en caso contrario.
;;
(defun literal-negativo-p (fbf) ...)

EJEMPLOS:
>> (literal-negativo-p '(¬ p))        ; T
>> (literal-negativo-p '(¬ T))        ; NIL
>> (literal-negativo-p '(¬ NIL))      ; NIL
>> (literal-negativo-p '(¬ =>))       ; NIL
>> (literal-negativo-p 'p)            ; NIL
>> (literal-negativo-p '((¬ p)))      ; NIL
>> (literal-negativo-p '(¬ (v p q)))  ; NIL
```

3. Escriba código LISP para determinar si una FBF es un literal.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; EJERCICIO 3
;; Predicado para determinar si una FBF es un literal
;;
;; RECIBE   : FBF en formato prefijo
;; EVALÚA A : T si FBF es un literal, NIL en caso contrario.
;;
(defun literal-p (fbf) ...)
```

```
EJEMPLOS:
>> (literal-p 'p)
>> (literal-p '(¬ p))
evalúan a T
>> (literal-p '(p))
>> (literal-p '(¬ (v p q)))
evalúan a  NIL
```

4. Escriba una función LISP para extraer todos los símbolos atómicos a partir de una FBF en cualquier tipo de formato.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; EJERCICIO 4
;;
;; RECIBE   : fórmula en cualquier formato (expr)
;; EVALÚA A : Lista de símbolos atómicos (sin repeticiones)
;;            utilizados en la fórmula bien formada. El orden en la
lista no es
;;            relevante.
;;
(defun extrae-simbolos (expr) ...)

EJEMPLOS:
(extrae-simbolos 'A)                          ; (A)
(extrae-simbolos '(H <=> (¬ H)))              ; (H)
(extrae-simbolos '((A <=> (¬ H)) ^ (P <=> (A ^ H)) ^ (H <=> P)))
;;(A P H) [En cualquier orden]

(extrae-simbolos '((<=> A  (¬ H)) (<=> P  (^ A  H)) (<=> H  P)))
;; (A P H) [En cualquier orden]
```

Representación de las fórmulas en forma FNC.

- **Cláusula** (cláusula disyuntiva): disyunción de literales.
- **Forma normal conjuntiva (FNC):** Fórmula bien formada que consiste en una conjunción de cláusulas disyuntivas. Es decir, es una conjunción entre disyunciones de literales.

---

5. Escriba código LISP para determinar si una FBF en formato prefijo es una cláusula (disyuntiva).

DEFINICIÓN: Una cláusula es una disyunción de literales, es decir, debería tener un formato

```
'(v lit1 lit2 ... litn)

CASOS ESPECIALES:      '(v ) disyunción vacía (valor de verdad False)
                       '(v lit1)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; EJERCICIO 5
;; Predicado para determinar si una FBF es una cláusula
;;
;; RECIBE   : FBF en formato prefijo
;; EVALÚA A : T si FBF es una cláusula, NIL en caso contrario.
;;
(defun clausula-p (fbf) ...)
```

```
EJEMPLOS:
(clausula-p '(v )             ; T
(clausula-p '(v p))           ; T
(clausula-p '(v (¬ r)))       ; T
(clausula-p '(v p q (¬ r) s)) ; T
(clausula-p 'p)                     ; NIL
(clausula-p '(¬ p))                 ; NIL
(clausula-p NIL)                    ; NIL
(clausula-p '(p))                   ; NIL
(clausula-p '((¬ p)))               ; NIL
(clausula-p '(^ a b q (¬ r) s))     ; NIL
(clausula-p '(v (^ a b) q (¬ r) s)) ; NIL
(clausula-p '(¬ (v p q)))           ; NIL
```

6. Escriba código LISP para determinar si una FBF en formato prefijo está en FNC.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; EJERCICIO 6
;; Predicado para determinar si una FBF está en FNC
;;
;; RECIBE   : FBF en formato prefijo
;; EVALÚA A : T si FBF está en FNC con conectores,
;;            NIL en caso contrario.
```

DEFINICIÓN: Una FNC es una conjunción de cláusulas, es decir, debería tener un formato

```
            '(^ K1 K2 ... Kn)

CASOS ESPECIALES:
      '(^)      conjunción vacía (valor de verdad True)
      '(^ (v )) conjunción con cláusula vacía (valor de verdad False)
      '(^ (v K1))

EJEMPLOS:
(FNC-p '(^ (v a  b c) (v q r) (v (¬ r) s) (v a b))) ; T
(FNC-p '(^))                                        ; T
(FNC-p '(¬ p))                                      ; NIL
(FNC-p '(^ a b q (¬ r) s))                          ; NIL
(FNC-p '(^ (v a b) q (v (¬ r) s) a b))              ; NIL
(FNC-p '(v p q (¬ r) s))                            ; NIL
(FNC-p '(^ (v a b) q (v (¬ r) s) a b))              ; NIL
(FNC-p '(^ p))                                      ; NIL
(FNC-p NIL)                                         ; NIL
(FNC-p '((¬ p)))                                    ; NIL
(FNC-p '(p))                                        ; NIL
(FNC-p '(^ (p)))                                    ; NIL
(FNC-p '((p)))                                      ; NIL
(FNC-p '(^ a b q (r) s))                            ; NIL
(FNC-p '(^ (v a  (v b c)) (v q r) (v (¬ r) s) a b)) ; NIL
(FNC-p '(^ (v a (^ b c)) (^ q r) (v (¬ r) s) a b))  ; NIL
(FNC-p '(¬ (v p q)))                                ; NIL
(FNC-p '(v p q (r) s))                              ; NIL
```