## 1.1 Similitud coseno recursiva

### 1.1.1 Pseudocódigo

* Entrada: vectores x, y
* Salida: la similitud coseno de de los vectores, calculado recursivo
* Procesamiento: Si los vectores son validos (no nil y no solo 0s) calcula el producto de los vectores recursivo. Después divide el producto por el producto de los raíces de los cuadrados de los vectores.

### 1.1.2 Ejemplos

(sc−rec nil nil) −> nil

(sc−rec (0 0) (1 2)) −> nil

(sc−rec (1 1) (1 1)) −> 1

(sc−rec (−1 −1) (1 1)) −> −1

(sc−rec (1 0) (0 1)) −> 0

(sc−rec (5 7) (4 3)) −> 0.95

### 1.1.3 Comentarios

* + A causa de los errores de redondeo, puede ser que los resultados no sean exactos (por ejemplo 1.0000001 en lugar de 1)
  + La recursión consiste en multiplicar los primeros dos elementes de cada vector y recurso en el resto hasta que no hay mas y en el camino de vuelta suma los productos
  + Cuando un vector es NIL o está constituido solo por ceros, nos devuelve el valor NIL.
  + Los cuadrados son calculado por el producto de el vector consigo mismo.

## 1.2 Similitud coseno mediante mapcar

### 1.2.1 Pseudocódigo

* Entrada: vectores x, y
* Salida: la similitud coseno de los vectores, calculado con mapcar
* Procesamiento: Si los vectores son validos (ni es NIL ni está compuesto solo de ceros) calcula el producto de los vectores con mapcar. Después divide el producto por el producto de los raíces de los vectores al cuadrado.

### 1.2.2 Ejemplos

Como recursivo

### **1.2.3 Comentarios**

* A causa de los errores de redondeo, puede ser que los resultados no sean exactos (por ejemplo 1.0000001 en lugar de 1)
* El producto de los vectores se calcula por una reducción con ’+’ sobre un mapcar que multiplica todas las parejas de los vectores
* Cuando un vector es NIL o está constituido solo por ceros, nos devuelve el valor NIL.
* Los cuadrados son calculado por el producto de el vector consigo mismo.

## 1.3 sc-conf

### 1.3.1 Pseudocódigo

* Entrada: vector cat, vector de vectores vs, nivel de confianza conf.
* Salida: Vectores cuya similitud a cat es superior a conf, ordenados.
* Procesamiento: Por cada vector de vs, calcula la similitud coseno con cat, después excluye los vectores con una similitud coseno inferior al nivel de confianza y ordena los vectores de mayor a menor similitud de coseno.

### **1.3.2 Ejemplos**

(sc−conf ’(1 2 3) ’((1 2 3) (0 1 0) (4 5 6)) 0.7) −> (0.97463 0.99999)

### 1.3.3 Comentarios

• Como en los casos anteriores, errores de redondeo son posibles.

## 1.4 sc classifier

### 1.4.1 Pseudocódigo

• Entrada: vector de vectores cats, texts; función func

• Salida: Por cada texto, la categoría que tiene la similitud coseno máximo y ese valor

• Procesamiento: Por cada vector de texto, calcula todas las scs con la funcion indicada con las categorías y devuelve el identificador y el resultado de la categoría con sc máxima.

### 1.4.2 Ejemplos

sc-classifier '((0 1 2 3 4) (1 3 2 2 0) (2 3 5 8 0)) '((0 1 2 3 4) (1 3 2 1 0) (2 1 1 1 1)) 'sc-mapcar) -> ((0 . 0.99999994) (1 . 0.97230554) (0 . 0.9128709))

### 1.4.3 Comentarios

Como se puede observar, calcular las scs es mas rápido con mapcar, pero necesita mas espacio.

## Ejercicio 2.1

### 2.1.1 Pseudocódigo

* Entrada: funciona f, extremo bajo a, extremo alto b, tolerancia tol
* Salida: el raiz de la funcion entre a y b cuando existe, nil si no
* Procesiamento:
  + Cuando no hay un raiz entre a y b, volver nil
  + Cuando f((a+b) /2) es 0 o la diferencia entre a y b es mas bajo que tol volver (a+b) / 2
  + Si no, busco entre el interval que tiene la raiz

### 2.1.2 Ejemplos

(bisect #'(lambda (x) (sin (\* 6.26 x))) 0.1 0.7 0.001) -> 0.501662

(bisect #'(lambda (x) (- (\* 2 x) 3)) 1.1 2.1 0.001) -> 1.4999022

(bisect #'(lambda (x) (- (\* 2 x) 3)) 1.25 1.75 0.001) -> 1.5

(bisect #'(lambda (x) (sin (\* 6.26 x))) 0.0 0.7 0.001) -> nil

### 2.1.3 Comentarios

A causa de los errores de redondeo, puede ser que los resultados no sean exactos (por ejemplo 1.0000001 en lugar de 1)

## Ejercicio 2.2

### 2.2.1 Pseudocódigo

* Entrada: funciona f, listo de intervalos lst, tolerancia tol
* Salida: Listo de las raizes entre los intervalos
* Procesamiento: Por todos los intervalos, si tienen una raíz, la raíz

### 2.2.2 Ejemplos

(allroot #'(lambda(x) (sin (\* 6.28 x))) '(0.25 0.75 1.25 1.75 2.25) 0.0001) -> (0.50027466 1.0005188 1.5007629 2.001007)

(allroot #'(lambda(x) (- (\* 2 x) 3)) '(0.25 0.75 1.25 1.75 2.25) 0.0001) ->(1.5)

### 2.2.3 Comentarios

A causa de los errores de redondeo, puede ser que los resultados no sean exactos (por ejemplo 1.0000001 en lugar de 1)

## Ejercicio 2.3

### 2.3.1 Pseudocódigo

* Entrada: funciona f, extremo bajo a, extremo alto b, exponent de 2 para segregar el intervalo, tolerancia tol
* Salida: Listo de las raizes entre los intervalos
* Procesamiento: crear una lista de los intervalos, despues aplicar ‘allroot’ a la lista

### 2.3.2 Ejemplos

(allind #'(lambda(x) (sin (\* 6.28 x ))) 0.1 2.25 1 0.0001) -> nil

(allind #'(lambda(x) (sin (\* 6.28 x ))) 0.1 2.25 2 0.0001) -> (0.50027096 1.000503 1.5007349 2.0010324)

### 2.3.3 Comentarios

A causa de los errores de redondeo, puede ser que los resultados no sean exactos (por ejemplo 1.0000001 en lugar de 1)

## Ejercicio 3.1

### 3.1.1 Pseudocódigo

* Entrada: elemento e, lista lst
* Salida: una lista de todos elementos de lst combinado con e
* Procesamiento: Por cada elemento de la lista, crear una lista consistente de lo y elemento e

### 2.3.2 Ejemplos

(combine-elt-lst 'a nil) -> NIL

(combine-elt-lst 'a '(1 2 3)) -> ((A 1) (A 2) (A 3))

(combine-elt-lst '(a b c) '(1 2 3)) -> ((A B C 1) (A B C 2) (A B C 3))

### 2.3.3 Comentarios

La solucion esta bastante feo para manajar correctamente con listos por elt tambien

## Ejercicio 3.2

### 3.2.1 Pseudocódigo

* Entrada: lista lst1, lista lst2
* Salida: una lista de todos elementos de lst1 combinado todos de lst2
* Procesamiento: Por cada elemento de la lista 1, aplicar combine-lst-lst con eso y lista 2

### 3.2.2 Ejemplos

(combine-lst-lst NIL '(a b c)) -> NIL

(combine-lst-lst '(a b c) '(1 2)) -> ((A 1) (A 2) (B 1) (B 2) (C 1) (C 2))

### 3.2.3 Comentarios

-

## Ejercicio 3.3

### 3.2.1 Pseudocódigo

* Entrada: lista de listos lstolsts
* Salida: todas las posibles disposiciones de elementos pertenecientes a N listas de forma que en cada disposición aparezca únicamente un elemento de cada lista:
* Procesamiento: Recursivo, aplica combine-lst-lst con todos los elementos acumulados y el primer elemento del resto de lstolsts

### 3.2.2 Ejemplos

(combine-list-of-lsts ’(() (+ -) (1 2 3 4))) -> NIL

(combine-list-of-lsts ’((1 2 3 4))) -> ((1) (2) (3) (4))

((A + 1) (A + 2) (A + 3) (A + 4) (A - 1) (A - 2) (A - 3) (A - 4) (B + 1) (B + 2) (B + 3) (B + 4) (B - 1)

(B - 2) (B - 3) (B - 4) (C + 1) (C + 2) (C + 3) (C + 4) (C - 1) (C - 2) (C - 3) (C - 4))

### 3.2.3 Comentarios

-

Ejercicio 4.1.1

1. Pseudocódigo

* Entrada: expresión x
* Salida: T si x es un literal positivo, NIL en caso de que no lo sea
* Procesamiento: dada la expresión x evalúa si es un literal positivo devolviendo si lo es o no.

1. Ejemplos

(print (positive-literal-p 'p)) −> T

(print (positive-literal-p T)) −> NIL

(print (positive-literal-p NIL)) −> NIL

(print (positive-literal-p '¬)) −> NIL

(print (positive-literal-p '=>)) −> NIL

(print (positive-literal-p '(p))) −> NIL

(print (positive-literal-p '(¬ p))) −> NIL

(print (positive-literal-p '(¬ (v p q)))) −> NIL

1. Comentarios

* Los literales positivos son aquellos que representan valores de verdad distintos de T y NIL y de los que representan conectores, así mismo si un literal positivo está precedido por el conector de negación pasa a ser un literal negativo.

Ejercicio 4.1.2

1. Pseudocódigo

* Entrada: expresión x
* Salida: T si x es un literal negativo, NIL en caso de que no lo sea
* Procesamiento: dada la expresión x evalúa si es un literal negativo devolviendo si lo es o no.

1. Ejemplos

(print (negative-literal-p '(¬ p))) −> T

(print (negative-literal-p NIL)) −> NIL

(print (negative-literal-p '¬)) −> NIL

(print (negative-literal-p '=>)) −> NIL

(print (negative-literal-p '(p))) −> NIL

(print (negative-literal-p '((¬ p)))) −> NIL

(print (negative-literal-p '(¬ T))) −> NIL

(print (negative-literal-p '(¬ NIL))) −> NIL

(print (negative-literal-p '(¬ =>))) −> NIL

(print (negative-literal-p 'p)) −> NIL

(print (negative-literal-p '((¬ p)))) −> NIL

(print (negative-literal-p '(¬ (v p q)))) −> NIL

1. Comentarios

* Los literales negativos son aquellas listas cuyo primer elemento es el conector negativo y cuyo segundo elemento es un literal negativo. Así cualquier conector devuelve el valor NIL así como los literales positivos.

Ejercicio 4.1.3

1. Pseudocódigo

* Entrada: expresión x
* Salida: T si x es un literal, NIL en caso de que no lo sea
* Procesamiento: dada la expresión x evalúa si es un literal devolviendo si lo es o no.

1. Ejemplos

(print (literal-p 'p)) −> T

(print (literal-p '(¬ p))) −> T

(print (literal-p '(p))) −> NIL

(print (literal-p '(¬ (v p q)))) −> NIL

1. Comentarios

* Se ha usado una función que aúna los valores positivos para la función que determina los literales positivos y para la que determina los literales negativos. Por lo que todos los literales están incluidos tanto positivos como negativos.

Ejercicio 4.1.4

1)

1. Pseudocódigo

* Entrada: expresión x
* Salida: T si x está en formato prefijo, NIL en caso de que no lo esté
* Procesamiento: dada la expresión x evalúa si está en formato prefijo devolviendo el resultado.

1. Ejemplos

(print (wff-prefix-p '(v))) −> T

(print (wff-prefix-p '(^)))−> T

(print (wff-prefix-p '(v A))) −> T

(print (wff-prefix-p '(^ (¬ B)))) −> T

(print (wff-prefix-p '(v A (¬ B)))) −> T

(print (wff-prefix-p '(v (¬ B) A ))) −> T

(print (wff-prefix-p '(^ (V P (=> A (^ B (¬ C) D))) (^ (<=> P (¬ Q)) P) E))) −> T

(print (wff-prefix-p 'NIL)) −> NIL

(print (wff-prefix-p '(¬))) −> NIL

(print (wff-prefix-p '(=>))) −> NIL

(print (wff-prefix-p '(<=>))) −> NIL

(print (wff-prefix-p '(^ (V P (=> A ( B ^ (¬ C) ^ D))) (^ (<=> P (¬ Q)) P) E))) −> NIL

1. Comentarios

* NIL no es un FBF en formato prefijo
* Un literal es FBF en formato prefijo
* Las listas cuyo primer elemento es un conector unario deben tener la estructura (<conector> FBF)
* Las listas cuyo primer elemento es un conector binario deben tener estructura (<conector> FBF1 FBF2)
* Las listas cuyo primer elemento es un conector enario deben tener por operandos FBF y el segundo operando podría estar vacío.

2)

1. Pseudocódigo

* Entrada: expresión x
* Salida: T si x está en formato infijo, NIL en caso de que no lo esté
* Procesamiento: dada la expresión x evalúa si está en formato infijo devolviendo el resultado.

1. Ejemplos

(print (wff-infix-p 'a)) −> T

(print (wff-infix-p '(^))) −> T

(print (wff-infix-p '(v))) −> T

(print (wff-infix-p '(¬ c))) −> T

(print (wff-infix-p '(A ^ (v)))) −> T

(print (wff-infix-p '(A ^ (p v q)))) −> T

(print (wff-infix-p '( a ^ b ^ (p v q) ^ (¬ r) ^ s))) −> T

(print (wff-infix-p '(A => B))) −> T

(print (wff-infix-p '(A => (B <=> C)))) −> T

(print (wff-infix-p '( B => (A ^ C ^ D)))) −> T

(print (wff-infix-p '( B => (A ^ C)))) −> T

(print (wff-infix-p '( B ^ (A ^ C)))) −> T

(print (wff-infix-p '((p v (a => (b ^ (¬ c) ^ d))) ^ ((p <=> (¬ q)) ^ p ) ^ e))) −> T

(print (wff-infix-p nil)) −> NIL

(print (wff-infix-p '(a ^))) −> NIL

(print (wff-infix-p '(^ a))) −> NIL

(print (wff-infix-p '(A ^ v))) −> NIL

(print (wff-infix-p '(c ¬))) −> NIL

(print (wff-infix-p '(c ¬ c))) −> NIL

(print (wff-infix-p '(a))) −> NIL

(print (wff-infix-p '((a)))) −> NIL

(print (wff-infix-p '((a) b))) −> NIL

(print (wff-infix-p '(^ a b q (¬ r) s))) −> NIL

(print (wff-infix-p '( B => A C))) −> NIL

(print (wff-infix-p '( => A))) −> NIL

(print (wff-infix-p '(A =>))) −> NIL

(print (wff-infix-p '(A => B <=> C))) −> NIL

(print (wff-infix-p '( B => (A ^ C v D)))) −> NIL

(print (wff-infix-p '( B ^ C v D ))) −> NIL

(print (wff-infix-p '((p v (a => e (b ^ (¬ c) ^ d))) ^ ((p <=> (¬ q)) ^ p ) ^ e))) −> NIL

1. Comentarios

* NIL no es un FBF en formato infijo
* Un literal es FBF en formato infijo
* Las listas cuyo primer elemento es un conector unario deben tener la estructura (<conector> FBF) para estar en formato infijo
* Las listas cuyo primer elemento es un conector binario deben tener estructura (FBF1 <conector> FBF2) para estar en formato infijo.

Ejercicio 4.1.5

1. Pseudocódigo

* Entrada: FBF en formato infijo
* Salida: FBF en formato prefijo
* Procesamiento: Dada una FBF en formato infijo transformarla y devolverla en formato prefijo, cambiando la estructura de la FBF.

1. Ejemplos

(print (infix-to-prefix nil)) −> NIL

(print (infix-to-prefix 'a)) −> a

(print (infix-to-prefix '((a)))) −> NIL

(print (infix-to-prefix '(a))) −> NIL

(print (infix-to-prefix '(((a))))) −> NIL

(print (infix-to-prefix '(a v p)))

(print (infix-to-prefix '(a v p v b)))

(print (infix-to-prefix '(a v p v b v x)))

(print (infix-to-prefix '(a v p v b v x v y)))

(print (equal (prefix-to-infix (infix-to-prefix '((p v (a => (b ^ (¬ c) ^ d))) ^ ((p <=> (¬ q)) ^ p) ^ e)) )

'((P V (A => (B ^ (¬ C) ^ D))) ^ ((P <=> (¬ Q)) ^ P) ^ E)))

(print (equal (infix-to-prefix '((p v (a => (b ^ (¬ c) ^ d))) ^ ((p <=> (¬ q)) ^ p) ^ e))

'(^ (V P (=> A (^ B (¬ C) D))) (^ (<=> P (¬ Q)) P) E)))

(print (equal (infix-to-prefix '(¬ ((¬ p) v q v (¬ r) v (¬ s))))

'(¬ (V (¬ P) Q (¬ R) (¬ S)))))

(print (equal (infix-to-prefix

(prefix-to-infix

'(V (¬ P) Q (¬ R) (¬ S))))

'(V (¬ P) Q (¬ R) (¬ S))))

(print (equal (infix-to-prefix

(prefix-to-infix

'(¬ (V (¬ P) Q (¬ R) (¬ S)))))

'(¬ (V (¬ P) Q (¬ R) (¬ S)))))

(print (equal (infix-to-prefix 'a) 'A))

(print (equal (infix-to-prefix '((p v (a => (b ^ (¬ c) ^ d))) ^ ((p <=> (¬ q)) ^ p) ^ e))

'(^ (V P (=> A (^ B (¬ C) D))) (^ (<=> P (¬ Q)) P) E)))

(print (equal (infix-to-prefix '(¬ ((¬ p) v q v (¬ r) v (¬ s))))

'(¬ (V (¬ P) Q (¬ R) (¬ S)))))

(print (equal (infix-to-prefix (prefix-to-infix '(^ (v p (=> a (^ b (¬ c) d)))))) '(v p (=> a (^ b (¬ c) d)))))

(print (equal (infix-to-prefix (prefix-to-infix '(^ (^ (<=> p (¬ q)) p ) e))) '(^ (^ (<=> p (¬ q)) p ) e)))

(print (equal (infix-to-prefix (prefix-to-infix '( v (¬ p) q (¬ r) (¬ s)))) '( v (¬ p) q (¬ r) (¬ s))))

(print (equal (infix-to-prefix '(p v (a => (b ^ (¬ c) ^ d)))) '(V P (=> A (^ B (¬ C) D)))))

(print (equal (infix-to-prefix '(((P <=> (¬ Q)) ^ P) ^ E)) '(^ (^ (<=> P (¬ Q)) P) E)))

(print (equal (infix-to-prefix '((¬ P) V Q V (¬ R) V (¬ S))) '(V (¬ P) Q (¬ R) (¬ S))))

1. Comentarios

Utilizamos recursividad para pasar los operadores de infijo a prefijo.

Debemos comprobar si son literales o listas para poder operar, por lo que hacemos las operaciones necesarias primero.

Ejercicio 4.1.6

1. Pseudocódigo

* Entrada: FBF en formato prefijo
* Salida: T si FBF es una cláusula ( disyunción de literales), NIL en caso de que no lo sea.
* Procesamiento: Dada una FBF en formato prefijo comprobar si tiene la forma '(V lit1 lit 2 … litn) siendo lit1, lit 2… litn literales. Devolver el resultado de la comprobación.

1. Ejemplos

(print (clause-p '(v))) −> T

(print (clause-p '(v p))) −> T

(print (clause-p '(v (¬ r)))) −> T

(print (clause-p '(v p q (¬ r) s))) −> T

(print (clause-p NIL)) −> NIL

(print (clause-p 'p)) −> NIL

(print (clause-p '(¬ p))) −> NIL

(print (clause-p NIL)) −> NIL

(print (clause-p '(p))) −> NIL

(print (clause-p '((¬ p)))) −> NIL

(print (clause-p '(^ a b q (¬ r) s))) −> NIL

(print (clause-p '(v (^ a b) q (¬ r) s))) −> NIL

(print (clause-p '(¬ (v p q)))) −> NIL

1. Comentarios

* Las cláusulas vacías no son una disyunción de literales, aunque el valor devuelto sea T
* Las cláusulas de un solo literal no son una disyunción de literales aunque el valor devuelto sea T

Ejercicio 4.1.7

1. Pseudocódigo

* Entrada: FBF en formato prefijo
* Salida: T si FBF es una FNC ( disyunción de literales), NIL en caso de que no lo sea.

Procesamiento: Dada una FBF es formato prefijo comprobar si tiene la forma '(∧ K1 K2 … Kn) siendo K1, K2 2… Kn cláusulas. Devolver el resultado de la comprobación.

1. Ejemplos

(print (cnf-p '(^ (v a b c) (v q r) (v (¬ r) s) (v a b)))) −> T

(print (cnf-p '(^ (v a b (¬ c)) ))) −> T

(print (cnf-p '(^ ))) −> T

(print (cnf-p '(^(v )))) −> T

(print (cnf-p '(¬ p))) −> NIL

(print (cnf-p '(^ a b q (¬ r) s))) −> NIL

(print (cnf-p '(^ (v a b) q (v (¬ r) s) a b))) −> NIL

(print (cnf-p '(v p q (¬ r) s))) −> NIL

(print (cnf-p '(^ (v a b) q (v (¬ r) s) a b))) −> NIL

(print (cnf-p '(^ p))) −> NIL

(print (cnf-p '(v ))) −> NIL

(print (cnf-p NIL)) −> NIL

(print (cnf-p '((¬ p)))) −> NIL

(print (cnf-p '(p))) −> NIL

(print (cnf-p '(^ (p)))) −> NIL

(print (cnf-p '((p)))) −> NIL

(print (cnf-p '(^ a b q (r) s))) −> NIL

(print (cnf-p '(^ (v a (v b c)) (v q r) (v (¬ r) s) a b))) −> NIL

(print (cnf-p '(^ (v a (^ b c)) (^ q r) (v (¬ r) s) a b))) −> NIL

(print (cnf-p '(¬ (v p q)))) −> NIL

(print (cnf-p '(v p q (r) s))) −> NIL

1. Comentarios

* La conjunción vacía es una FNC y el valor devuelto es T
* La conjunción de una única cláusula no es una FNC y el valor devuelto es T
* La conjunción con una única cláusula formada por un único literal no es una FNC

Ejercicio 4.2.1

1. Pseudocódigo

* Entrada: FBF en formato prefijo
* Salida: la misma FBF en formato prefijo sin conectores bicondicionales.
* Procesamiento: dada una FBF en formato prefijo sustituir los conectores bicondicionales siguiendo la regla: (fbf1 <=> fbf2) ≡ ((fbf1 => fbf2) ∧ (fbf2 => fbf1). Y devolver la fbf tras la sustitución.

1. Ejemplos

(print (eliminate-biconditional '(<=> p (v q s p) ))) −> (^ (=> P (v Q S P)) (=> (v Q S P) P))

(print (eliminate-biconditional '(<=> (<=> p q) (^ s (¬ q))))) −> (^ (=> (^ (=> P Q) (=> Q P)) (^ S (¬ Q))) (=> (^ S (¬ Q)) (^ (=> P Q) (=> Q P))))

1. Comentarios

* Eliminamos de forma recursiva el conector bicondicional cuando lo hay y lo transformamos en una lista de listas con condicionales simples. Al ser en formato prefijo nos resulta más simple.

Ejercicio 4.2.2

1. Pseudocódigo

* Entrada: FBF en formato prefijo sin conectores bicondicionales
* Salida: la misma FBF en formato prefijo sin conectores condicionales.
* Procesamiento: dada una FBF en formato prefijo sin conectores bicondicionales sustituir los conectores condicionales siguiendo la regla: (fbf1 => fbf2) ≡ ((¬fbf1) V fbf2). Y devolver la fbf tras la sustitución.

1. Ejemplos

(print (equal (eliminate-conditional '(=> p q)) −> '(V (¬ P) Q)))

(print (equal (eliminate-conditional '(=> p (v q s p))) −> '(V (¬ P) (V Q S P))))

(print (equal (eliminate-conditional '(=> (=> (¬ p) q) (^ s (¬ q)))) −> '(V (¬ (V (¬ (¬ P)) Q)) (^ S (¬ Q)))))

1. Comentarios

* El codigo es muy similar al anterior, solo que esta vez hacemos un or de cada FBF negada.

Ejercicio 4.2.3

1. Pseudocódigo

* Entrada: FBF en formato prefijo sin conectores bicondicionales ni condicionales
* Salida: la misma FBF en formato prefijo sin negación, a menos que corresponda a literales negativos
* Procesamiento: dada una FBF en formato prefijo sin conectores bicondicionales ni condicionales sustituir los operadores de negación siguiendo las siguientes reglas:

(¬ (fbf1 ∨ fbf2 ∨ fbf3)) ≡ ((¬ fbf1) ∧ (¬ fbf2) ∧ (¬ fbf3))

(¬ (fbf1 ∧ fbf2 ∧ fbf3)) ≡ ((¬ fbf1) ∨ (¬ fbf2) ∨ (¬ fbf3))

(¬ (¬ c1)) ≡ c1

Devolviendo una FBF equivalente en la que los operadores negativos solo corresponden a literales negativos.

1. Ejemplos

(print (reduce-scope-of-negation 'r))

(print (reduce-scope-of-negation '(¬ r)))

(print (reduce-scope-of-negation '(v p)))

(print (reduce-scope-of-negation '(v p r)))

(print (reduce-scope-of-negation '(¬ (v p))))

(print (reduce-scope-of-negation '(¬ (v p r))))

(print (reduce-scope-of-negation '(¬ (v p (¬ q) r))))

(print (equal (reduce-scope-of-negation '(¬ (v p (¬ q) r))) −> '(^ (¬ P) Q (¬ R))))

(print (equal (reduce-scope-of-negation '(¬ (^ p (¬ q) (v r s (¬ a))))) −> '(V (¬ P) Q (^ (¬ R) (¬ S) A))))

(print (equal (reduce-scope-of-negation '(¬ (v fbf1 fbf2 fbf3))) −> '(^ (¬ fbf1) (¬ fbf2) (¬ fbf3))))

(print (reduce-scope-of-negation '(¬ (v fbf1 fbf2 fbf3))))

(print (equal (reduce-scope-of-negation '(¬ (^ fbf1 fbf2 fbf3))) −> '(v (¬ fbf1) (¬ fbf2) (¬ fbf3))))

(print (reduce-scope-of-negation '(¬ (^ fbf1 fbf2 fbf3))))

(print (reduce-scope-of-negation '(v fbf1 fbf3)))

(print (reduce-scope-of-negation '(¬ (v fbf1 fbf3))))

(print (reduce-scope-of-negation '(^ (¬ (v fbf1 fbf3)) fbf2 fbf3)))

(print (reduce-scope-of-negation '(¬ (^ (¬ (v fbf1 fbf3)) fbf2 fbf3))))

(print (reduce-scope-of-negation '(¬ (¬ c1))))

1. Comentarios

* En este ejercicio nos hemos apoyado en la funcion exchange-and-or facilitada para simplificar el codigo de nuestra funcion.
* Recursivamente las clausulas van viendo eliminada su negacion y sustituida por los conectores opuestos, siguiendo las leyes de Morgan.

Ejercicio 4.2.4

1. Pseudocódigo

* Entrada: FBF en formato prefijo sin conectores bicondicionales ni condicionales y en la que la negación aparece únicamente en literales negativos
* Salida: la misma FBF en formato prefijo FNC.
* Procesamiento: dada una FBF en formato prefijo sin conectores bicondicionales ni condicionales y en la que la negación aparece únicamente en literales negativos la transforma en FNC distribuyendo y agrupando los conectores de conjunción y de disyunción, obteniendo una conjunción de disyunciones de literales, para ello se aplica la siguiente regla:

((c1 ∧ c2) ∨ (c3 ∨ c4)) ≡ ((c1 ∧ c2) ∨ c3 ∨ c4) ≡ ((c1 ∨ c3 ∨ c4) ∧ (c2 ∨ c3 ∨ c4))

1. Ejemplos

(print (cnf NIL)) −> NIL

(print (cnf 'a)) −> (^ (V A))

(print (cnf '(¬ a))) −> (^ (V (¬ A)))

(print (cnf '(V (¬ P) (¬ P)))) −> (^ (V (¬ P) (¬ P)))

(print (cnf '(V A))) −> (^ (V A))

(print (cnf '(^ (v p (¬ q)) (v k r (^ m n))))) −> (^ (V P (¬ Q)) (V K R M) (V K R N))

(print (cnf '(v (v p q) e f (^ r m) n (^ a (¬ b) c) (^ d s)))) −> (^ (V P Q E F R N A D) (V P Q E F R N A S) (V P Q E F R N (¬ B) D) (V P Q E F R N (¬ B) S) (V P Q E F R N C D) (V P Q E F R N C S) (V P Q E F M N A D) (V P Q E F M N A S) (V P Q E F M N (¬ B) D) (V P Q E F M N (¬ B) S) (V P Q E F M N C D)(V P Q E F M N C S))

(print

(cnf '(^ (^ (¬ y) (v r (^ s (¬ x)) (^ (¬ p) m (v c d)))(v (¬ a) (¬ b))) g))) −> (^ (V (¬ Y)) (V R S (¬ P)) (V R S M) (V R S C D) (V R (¬ X) (¬ P)) (V R (¬ X) M) (V R (¬ X) C D) (V (¬ A) (¬ B)) (V G))

1. Comentarios

Ejercicio 4.2.5

1. Pseudocódigo

* Entrada: Una FBF en FNC con conectores conjuncion y disyuncion
* Salida: Lista de listas de literales sin conectores
* Procesamiento: dada una FBF en FNC elimina los conectores de conjuncion y disyuncion de forma que se sobreentienden las conjunciones de cláusulas y las disyunciones de literales.

1. Ejemplos

(print (eliminate-connectors '(^ (v p (¬ q)) (v k r)))) → ((P (¬ Q)) (K R))  
(print (equal (eliminate-connectors '(^ (v p (¬ q)) (v k r))) '((P (¬ Q)) (K R))))  
(print (eliminate-connectors '(^ (v p (¬ q)) (v q (¬ a)) (v s e f) (v b))))  
(print (equal (eliminate-connectors '(^ (v p (¬ q)) (v q (¬ a)) (v s e f) (v b))) '((P (¬ Q)) (Q (¬ A)) (S E F) (B)))) → ((P (¬ Q)) (Q (¬ A)) (S E F) (B))

C) Comentarios

* Es una simple recursion de un mapcar para ir eliminando recursivamente los conectores.

Ejercicio 4.2.6

1. Pseudocódigo

* Entrada: Una FBF en notacion infija
* Salida: FNC sin conjunciones ni disyunciones
* Procesamiento: dada una FBF se transforma en una FNC eliminando los conectores existentes y dejandolo como una lista de listas.

1. Ejemplos

(print (wff-infix-to-cnf 'a))  
(print (wff-infix-to-cnf '(¬ a)))  
(print (wff-infix-to-cnf '( (¬ p) v q v (¬ r) v (¬ s))))  
(print (wff-infix-to-cnf '((p v (a => (b ^ (¬ c) ^ d))) ^ ((p <=> (¬ q)) ^ p) ^ e))) →   
 ((P (¬ A) B) (P (¬ A) (¬ C)) (P (¬ A) D) ((¬ P) (¬ Q)) (Q P) (P) (E))  
(print (equal (wff-infix-to-cnf '((p v (a => (b ^ (¬ c) ^ d))) ^ ((p <=> (¬ q)) ^ p) ^ e)) '((P (¬ A) B) (P (¬ A) (¬ C)) (P (¬ A) D) ((¬ P) (¬ Q)) (Q P) (P) (E))))

C) Comentarios

* En este ejercicio utilizamos todas las funciones que hemos creado en los ejercicios anteriores para recursivamente transformar ela FBF a FNC como lista de lista de literales.

Ejercicio 4.3.1

1. Pseudocódigo

* Entrada: Lista de literales
* Salida: Lista de los mismos literales pero sin repetidos
* Procesamiento: dada una lista de literales con la clausula disyuntiva implicita se eliminan los literales repetidos.

1. Ejemplos

(print (eliminate-repeated-literals '(a b (¬ c) (¬ a) a c (¬ c) c a)))

(print (equal (eliminate-repeated-literals '(a b (¬ c) (¬ a) a c (¬ c) c a)) '(B (¬ A) (¬ C) C A)))

C) Comentarios

* En este ejercicio buscamos los literales repetidos con una funcion lambda, los eliminamos y repetimos recursivamente con el resto de la lista.

Ejercicio 4.3.2

1. Pseudocódigo

* Entrada: Lista de clausulas FBF en FNC con clausulas repetidas
* Salida: Lista de las mismas clausulas pero sin repetidas
* Procesamiento: dada una lista de clausulas con la conjuncion disyuntiva implicita se eliminan las clausulas repetidas.

1. Ejemplos

(print (eliminate-repeated-clauses '(a a b c a)))

(print (eliminate-repeated-clauses '((a b) (a b))))

(print (eliminate-repeated-clauses '(((¬ a) c) (c (¬ a)) ((¬ a) (¬ a) b c b) (a a b) (c (¬ a) b b) (a b)))) → ((C (¬ A)) (C (¬ A) B) (A B))

(print '((C (¬ A)) (C (¬ A) B) (A B)))

(print (equal (eliminate-repeated-clauses '(((¬ a) c) (c (¬ a)) ((¬ a) (¬ a) b c b) (a a b) (c (¬ a) b b) (a b))) '((C (¬ A)) (C (¬ A) B) (A B)))) → semantically equal

C) Comentarios

* En este ejercicio nos hemos ayudado de funciones externas para simplificar el problema.
* Las funciones nos ayudan a comprobar si dos elementos de la lista de clausulas son iguales, si hay diferencias y en qué difieren.
* Para ver como difieren entre si comprobamos las distintas posibilidades de combinacion entre a b c y d.

Ejercicio 4.3.3

1. Pseudocódigo

* Entrada: Lista de de dos cláusulas
* Salida: La primera clausula si esta subsume a la segunda, NIL en caso de que no lo haga.
* Procesamiento: Compara ambas clausulas y si son iguales devuelve la primera y si no devuelve NIL.

1. Ejemplos

(print "subsume")

(print (subsume '(a) '(a b (¬ c))))

(print (equal (subsume '(a) '(a b (¬ c))) '((a)))) → ((a))

(print (subsume NIL '(a b (¬ c))))

(print (equal (subsume NIL '(a b (¬ c))) '(NIL))) → (NIL)

(print (subsume '(a b (¬ c)) '(a) ))

(print (equal (subsume '(a b (¬ c)) '(a)) NIL)) → NIL

C) Comentarios

* En este ejercicio simplemente llamamos a una funcion que comprueba si son diferentes y en caso de que lo sean devuelve nil, en caso contrario devuelve k1, que corresponde a la primera clausula.

Ejercicio 4.3.4

1. Pseudocódigo

* Entrada: Una FNC con clausulas subsumidas
* Salida: FBF en FNC sin las clausulas que estaban subsumidas en la cnf de la entrada
* Procesamiento: Elimina las clausulas subsumidas.

1. Ejemplos

(print "eliminiate subsumed clauses")

(print (eliminate-subsumed-clauses

'((a b c) (b c) (a (¬ c) b) ((¬ a) b) (a b (¬ a)) (c b a))))

(print '((A (¬ C) B) ((¬ A) B) (B C))) ;; el orden no es importante

(print (eliminate-subsumed-clauses

'((a b c) (b c) (a (¬ c) b) (b) ((¬ a) b) (a b (¬ a)) (c b a))))

(print '((B)))

C) Comentarios

* En este ejercicio utilizamos funciones auxiliares para encontras las clausulas que estan subsumidas entre si y poder eliminarlas recursivamente despues.

Ejercicio 4.3.5

1. Pseudocódigo

* Entrada: Una clausula para verificar si es tautologia o no
* Salida: True si es, NIL si no.
* Procesamiento: Comprueba si la clausula de entrada es una tautologia y devuelve true si lo es o nil si no lo es.

1. Ejemplos

(print (tautology-p '(A (¬ A))))

(print (equal (tautology-p '((¬ B) A C (¬ A) D)) 't)) → T

(print (equal (tautology-p '((¬ B) A C D)) nil)) → NIL

C) Comentarios

* En este ejercicio tenemos que evaluar cada literal independientemente para ver si la clausula completa es o no una tautologia, para ello utilizamos la funcion definida antes negative-literal-p y vamos procesando la clausula completa recursivamente hasta tener la evaluacion completa de la misma y devolver el resultado.

Ejercicio 4.3.6

1. Pseudocódigo

* Entrada: Una FBF en FNC
* Salida: La misma FBF en FNC pero sin tautologias
* Procesamiento: Recibe la cnf y elimina las tautologias llamando a la funcion del ejercicio anterior que las detecta.

1. Ejemplos

(print (eliminate-tautologies

'(((¬ b) a) (a (¬ a) b c) ( a (¬ b)) (s d (¬ s) (¬ s)) (a)))) → (((¬ B) A) (A (¬ B)) (A))

(print (eliminate-tautologies '((a (¬ a) b c)))) → NIL

C) Comentarios

* En este ejercicio simplemente llamamos a la funcion que reconoce tautologias en una clausula y elimina las mismas si y solo si el resultado de la llamada a esa funcion es true.

Ejercicio 4.3.7

1. Pseudocódigo

* Entrada: Una FBF en FNC
* Salida: La misma FBF en FNC pero sin tautologias, clausulas repetidas, literales ni clausulas subsumidas.
* Procesamiento: Simplifica la FBF eliminando primero literales repetidos en cada clausula, eliminando despues clausulas repetidas, eliminando tautologias y finalmente eliminando clausulas subsumidas

1. Ejemplos

(print (simplify-cnf '((a a) (b) (a) ((¬ b)) ((¬ b)) (a b c a) (s s d) (b b c a b))))

(print '((B) ((¬ B)) (S D) (A)))

C) Comentarios

* Este ejercicio sencillamente llama recursivamente a las funciones que hemos creado con anterioridad para eliminar tautologias, literales clausulas repetidas y subsumidas. El resultado es una FBf simplificada al maximo.

Ejercicio 4.4.1

1. Pseudocódigo

* Entrada: Una FBF en FNC simplificada y un literal positivo.
* Salida: Conjunto de clausulas lambda neutras de la FNC dada.
* Procesamiento: Construye el subconjunto de clausulas de cnf que no contienen el literar lambda ni el lambda negado.

1. Ejemplos

(print (extract-neutral-clauses 'p

'((p (¬ q) r) (p q) (r (¬ s) q) (a b p) (a (¬ p) c) ((¬ r) s))))

→ ((R (¬ S) Q) ((¬ R) S))

(print (null (extract-neutral-clauses 'r NIL))) → NIL

(print (equal (extract-neutral-clauses 'r '(NIL)) '(NIL))) → (NIL)

C) Comentarios

* Este ejercicio utiliza una funcion auxiliar que busca el literal dado en la clausula tanto en su forma positiva como negativa.
* Elimina recursivamente las apariciones para simplificar la FNC.

Ejercicio 4.4.2

1. Pseudocódigo

* Entrada: Una FBF en FNC simplificada y un literal positivo.
* Salida: Conjunto de clausulas lambda positivas de la FNC dada con el literal lambda
* Procesamiento: Construye el subconjunto de cláusulas de cnf que si contienen el literal lambda positivo.

1. Ejemplos

(print (equal (extract-positive-clauses 'p

'((p (¬ q) r) (p q) (r (¬ s) q) (a b p) (a (¬ p) c) ((¬ r) s)))

'((P (¬ Q) R) (P Q) (A B P))))

(print (null (extract-positive-clauses 'r NIL))) → NIL

(print (null (extract-positive-clauses 'r '(NIL)))) → NIL

C) Comentarios

* Este ejercicio utiliza una funcion auxiliar que comprueba que el literal dado no está en la clausula en su forma positiva.
* Elimina recursivamente las cláusulas en las que no está para simplificar la FNC y obtener únicamente las cláusulas que contienen el literal lambda.

Ejercicio 4.4.3

1. Pseudocódigo

* Entrada: Una FBF en FNC simplificada y un literal positivo.
* Salida: Conjunto de clausulas lambda negativas de la FNC dada con el literal lambda
* Procesamiento: Construye el subconjunto de clausulas de cnf que si contienen el literal lambda negativo.

1. Ejemplos

(print (equal (extract-negative-clauses 'p

'((p (¬ q) r) (p q) (r (¬ s) q) (a b p) (a (¬ p) c) ((¬ r) s)))

'((A (¬ P) C))))

(print (null (extract-negative-clauses 'r NIL))) → NIL

(print (null (extract-negative-clauses 'r '(NIL)))) → NIL

C) Comentarios

* En este ejercicio hemos hecho exactamente lo mismo que en el anterior pero esta vez con el lambda negativo.
* Este ejercicio utiliza una funcion auxiliar que comprueba que el literal dado no está en la clausula en su forma negativa.
* Elimina recursivamente las clausulas en las que no está para simplificar la FNC y obtener unicamente las clausulas que contienen el literal lambda en su forma negativa.

Ejercicio 4.4.4

1. Pseudocódigo

* Entrada: Dos cláusulas simplificadas y un literal positivo.
* Salida: Una sola cláusula resolvente sin literales repetidos
* Procesamiento: Dadas las dos cláusulas y el literal procesa y extrae el resolvente de ambas cláusulas y elimina los literales repetidos.

1. Ejemplos

(print "resolve on")

(print (null (resolve-on 'p '(a p) '(b p))))

(print (resolve-on 'p '(a p) '(b (¬ p))))

(print (resolve-on 'p '(a p) '(a (¬ p))))

(print (resolve-on 'p '((¬ a) p) '((¬ a) (¬ p))))

(print (resolve-on 'p '(a b (¬ c) p) '((¬ p) b a q r s))) → (((¬ C) B A Q R S))

(print (resolve-on 'p '(a b (¬ c) (¬ p)) '( p b a q r s))) → (((¬ C) B A Q R S))

(print (null (resolve-on 'p '(p) '((¬ p))))) → (NIL)

(print (null (resolve-on 'p NIL '(p b a q r s)))) → NIL

(print (null (resolve-on 'p NIL NIL))) → NIL

C) Comentarios

* Para este ejercicio hemos utilizado funciones auxiliares para comprobar si lambda es miembro de la cláusula y si es de forma positiva o negativa.
* Cuando encuentra los literales repetidos los elimina y extrae el resolvente de ambas cláusulas y hace la unión de las mismas.

Ejercicio 4.4.5

1. Pseudocódigo

* Entrada: FBF en FNC simplificada y un literal positivo.
* Salida: Conjunto de cláusulas RES de la FNC.
* Procesamiento: Dadas la FNC calcula su conjunto RES y elimina las cláusulas repetidas.

1. Ejemplos

(print (equal (build-RES 'p '(NIL)) '(nil))) → (NIL)

(print (equal (build-RES 'p '((p) ((¬ p)))) '(nil))) → (NIL)

(print (build-RES 'q '((p q) ((¬ p) q) (a b q) (p (¬ q)) ((¬ p) (¬ q)))))

(print '((P) ((¬ P) P) ((¬ P)) (B A P) (B A (¬ P))))

(print (build-RES 'p '((p q) (c q) (a b q) (p (¬ q)) (p (¬ q))))) → ((A B Q) (C Q))

C) Comentarios

* Para este ejercicio hemos utilizado funciones auxiliares programadas con anterioridad. Nos han servido las de extraccion de cláusulas neutrales, positivas y negativas.
* La funcion auxiliar que hemos creado sirve para simplificar el codigo de la funcion principal y simplemente llama recursivamente a la funcion que extrae el resolvente de dos cláusulas, lo cual nos sirve tambien en este ejercicio.

Ejercicio 4.5

1. Pseudocódigo

* Entrada: FBF en FNC simplificada
* Salida: T si cnf es SAT, NIL si cnf es UNSAT
* Procesamiento: Comprueba si una FNC es SAT calculando el resolvente para todos los átomos en la FNC

1. Ejemplos
2. C) Comentarios

(print (RES-SAT-p '((p) ((¬ q))))) → T

(print (RES-SAT-p

'((a b d) ((¬ p) q) ((¬ c) a b) ((¬ b) (¬ p) d) (c d (¬ a))))) → T

(print (RES-SAT-p

'(((¬ p) (¬ q) (¬ r)) (q r) ((¬ q) p) ((¬ q)) ((¬ p) (¬ q) r)))) → T

(print (null (RES-SAT-p '((p) ((¬ p)))))) → NIL

(print (null (RES-SAT-p

'(((¬ p) (¬ q) (¬ r)) (q r) ((¬ q) p) (p) (q) ((¬ r)) ((¬ p) (¬ q) r))))) ;;; NIL

* Para este ejercicio hemos utilizado funciones auxiliares.
* Same-atom-p comprueba que el literal sea igual dados dos literales independientemente del signo que le acompañe (esto es, si es negativo se considerara igual que si no lo es)
* get-literals extrae los literales de la cláusula.
* res-sat-aux es la funcion principal, ya que la usamos para simplificar el codigo de RES-SAT, ya que esta solo llama a funciones externas.
* res-sat-aux es la que comprueba si cnf es sat y devuelve true en ese caso, hace todas las comprobaciones de posibles errores, y utiliza funciones que hemos implementado antes para simplificar las cláusulas.

Ejercicio 4.6

1. Pseudocódigo

* Entrada: Dos FBF en formato infijo
* Salida: T si el primer argumento es consecuencia logica del segundo (w de wff), si no, devuelve NIL.
* Procesamiento: Comprueba que w sea consecuencia logica de wff

1. Ejemplos

(print (logical-consequence-RES-SAT-p '((p => q) ^ p) 'q)) → T

(print (null (logical-consequence-RES-SAT-p '((p => q) ^ p) '(¬q)))) → NIL

(print (logical-consequence-RES-SAT-p

'(((¬ p) => q) ^ (p => (a v (¬ b))) ^ (p => ((¬ a) ^ b)) ^ ( (¬ p) => (r ^ (¬ q))))

'(¬ a))) → T

C) Comentarios

* Este ejercicio basicamente utiliza funciones programadas con anterioridad como wff-infix-to-cnf para convertir a FNC la FBF y despues RES-SAT para determinar si son o no SAT.