

Deducción Natural

Dadas las nuevas reglas:

$$\frac{\Gamma, \Delta \neg \tau \vdash \tau}{\Gamma \vdash \Delta \tau} \Delta_i$$

$$\frac{\Gamma \vdash \Delta \tau \quad \Gamma \vdash \Delta \neg \tau}{\Gamma \vdash \tau} \Delta_e$$

Demostrar las siguientes fórmulas:

- a) $\vdash \tau \Rightarrow \Delta \tau$
- b) $\Delta \tau \vdash \Delta \neg \neg \tau$ (ojo, no se puede que $\neg \neg \neg \tau = \neg \tau$)
- c) $\Delta \tau, \Delta \neg \tau \vdash \sigma$ sugiere usar \perp_e y el ítem anterior.

$$\frac{\Gamma', \Delta \neg \neg \tau \vdash \tau}{\Gamma \vdash \Delta \neg \neg \tau} \Delta_i$$

$$\frac{\Gamma' \vdash \Delta \neg \tau \quad \Gamma' \vdash \Delta \tau}{\Gamma \vdash \Delta \neg \neg \tau} \Delta_e$$

$$\frac{\Gamma' \vdash \Delta \neg \neg \tau \vdash \tau}{\Gamma \vdash \Delta \neg \neg \tau} \Delta_i$$

$$\frac{\Gamma \vdash \Delta \neg \neg \tau \vdash \tau}{\Gamma \vdash \perp} \perp_e$$

$$\frac{\Gamma \vdash \Delta \neg \neg \tau \vdash \tau}{\Delta \vdash \Delta \neg \neg \tau} \Delta$$

$$\frac{\Gamma \vdash \Delta \neg \neg \tau \vdash \tau}{\Delta \vdash \Delta \neg \neg \tau} \Delta$$

$$\frac{\Gamma \vdash \Delta \neg \neg \tau \vdash \tau}{\Delta \vdash \Delta \neg \neg \tau} \Delta$$

$$\frac{\Gamma \vdash \Delta \neg \neg \tau \vdash \tau}{\Delta \vdash \Delta \neg \neg \tau} \Delta$$

b)

$$\frac{\Gamma \vdash \Delta \neg \neg \tau \vdash \tau}{\Delta \vdash \Delta \neg \neg \tau} \Delta$$

$$\frac{\Gamma \vdash \Delta \neg \neg \tau \vdash \tau}{\Delta \vdash \Delta \neg \neg \tau} \Delta$$

$$\frac{\Gamma \vdash \Delta \neg \neg \tau \vdash \tau}{\Delta \vdash \Delta \neg \neg \tau} \Delta$$

$$\frac{\Gamma \vdash \Delta \neg \neg \tau \vdash \tau}{\Delta \vdash \Delta \neg \neg \tau} \Delta$$

la refutación no exista, la función debe devolver False, o posiblemente True.

Ejercicio 3 (Deducción natural). Extendemos las fórmulas de la lógica de primer orden con un nuevo cuantificador $\exists x. \tau$ con las siguientes reglas de introducción y eliminación:

$$\frac{\Gamma, \neg \tau \{x := t\} \vdash \perp}{\Gamma \vdash \exists x. \tau} \exists_i$$

$$\frac{\Gamma \vdash \exists x. \tau \quad \Gamma \vdash \neg \tau \quad x \notin \text{fv}(\Gamma)}{\Gamma \vdash \perp} \exists_e$$

Demostrar que $\vdash (\exists x. P(f(x))) \Rightarrow \exists x. P(x)$.

Ayuda: el significado intuitivo del nuevo cuantificador es equivalente al del cuantificador existencial usual.

$$\frac{\Gamma'' \vdash \perp}{\Gamma'' \vdash \neg \neg \perp} \neg\neg_e$$

$$\frac{\Gamma'' \vdash \neg \neg \perp}{\Gamma'' \vdash \exists x. P(x)} \exists_i$$

$$\frac{\Gamma' \vdash \exists x. P(x)}{\Gamma' \vdash \neg \neg \exists x. P(x)} \neg\neg_e$$

$$\frac{\Gamma' \vdash \neg \neg \exists x. P(x)}{\Gamma' \vdash \exists x. P(x)} \exists_e$$

$$\frac{\Gamma \vdash \exists x. P(x)}{\Gamma \vdash \exists x. P(f(x))} \exists_i$$

$$\frac{\Gamma \vdash \exists x. P(f(x))}{\Gamma \vdash \exists x. P(x)} \exists_e$$

$$\frac{\Gamma \vdash \exists x. P(x)}{\vdash \exists x. P(x)} \text{PBC}$$

Ejercicio 1 (Programación funcional). Considerar la siguiente función:

```
foldl z f [] = z
foldl z f (x : xs) = f (x (foldl z f xs)) (foldl z f xs)
```

a) Dar el tipo de foldl.

b) Definir foldr sin usar recursión explícita, usando foldl.

c) Definir foldl sin usar recursión explícita, usando foldr.

$$b \rightarrow (a \rightarrow b \rightarrow b) \rightarrow [b \rightarrow a] \rightarrow b$$

foldr :: (a -> b -> b) -> b -> [a] -> b

foldr _ z [] = z

foldr f z (x:xs) = f x (foldr f z xs)

$$a) \text{Foldl } z f xs = \text{Foldl } (\lambda x. R \rightarrow f(x) R) R z xs$$

for que la función
 $f(x) R$
 mire con el const [1, 2, 3, 4]
 Entonces para
 se loren [Const 1, Const 2 ...]

$$b) \text{Foldr } f z xs = \text{Foldl } z (\lambda x. R \rightarrow f(x) R) (\text{map } (\text{const}) xs)$$

Ejercicio 3 (Programación lógica). Un árbol general es o bien el símbolo x o bien una lista de árboles generales.
 Por ejemplo, los siguientes son árboles generales:

$$x \quad [] \quad [x, x] \quad [[x, x], x, [x, x]] \quad [x, [x, x, [x, [], [x, x]], x], [x, []]]$$

Definir en Prolog un predicado arbolGeneral(-A) que genere todos los posibles árboles generales.

Ejercicio 1 (Razonamiento eucacional). Consideramos el tipo de datos de los árboles binarios

data AB a = Nil | Bin (AB a) a (AB a)

y las siguientes funciones:

```
(P0) post Nil = []
(P1) post (Bin i x d) = post d ++ post i ++ [x]
(M0) map f [] = []
(M1) map f (x : xs) = f x : map f xs
(A0) mapA f Nil = []
(A1) mapA f (Bin i x d) = mapA f d ++ mapA f i ++ mapA f x
(R0) rev ac [] = ac
(R1) rev ac (x : xs) = rev (x : ac) xs
```

Demostrar que para toda $f :: a \rightarrow b$ vale $\text{map } f . \text{post} = \text{rev } [] . \text{mapA } f$

Por Principio de Extensibilidad QVA

$\forall t :: AB \& \forall f :: a \rightarrow b . \text{map } f . \text{post } t = \text{new } [] . \text{mapA } f t$

Por inducción estructural en t QVA $P(t) = \text{new } [] . \text{mapA } f t$

El "ellos" hablan, regresión hacia que expanden

Caso Base $P(Nil) = \text{mapA } f . \text{post } Nil = \text{new } [] . \text{mapA } f . Nil$

$$\{P_0\} = \text{mapA } f [] \quad \{A_0\} = \text{new } [] []$$

$$\{R_0\} = [] \quad \{R_0\} = [] \checkmark$$

Caso Inductivo $P(Bin i R d) \quad \forall R :: a . (P(i) \wedge P(d)) \rightarrow P(Bin i R d)$

$P(Bin i R d) = \text{mapA } f . \text{post } (Bin i R d) = \text{new } [] . \text{mapA } f (Bin i R d)$

$$\{P1\} = \text{mapA } f (\text{post } d ++ \text{post } i ++ [x]) \quad \{A1\} = \text{new } [] ([F_x] ++ \text{mapA } f_c ++ \text{mapA } f_d)$$

$$\text{demo1} = \text{mapA } f (\text{post } d) ++ \text{mapA } f (\text{post } i) ++ \text{mapA } f [x] \quad \text{demo2}^{\text{new}} = \text{new } [] \text{ mapA } f_d ++ \text{new } [] \text{ mapA } f_i ++ \text{new } [] [F_x]$$

$$\{M1\} = \text{new } [] \text{ mapA } f_d ++ \text{new } [] \text{ mapA } f_i ++ \text{mapA } f [x] \rightarrow [x] = (x : []) \quad \{R1\} = \text{new } [] \text{ mapA } f_d ++ \text{new } [] \text{ mapA } f_i ++ \text{new } [] [F_x] []$$

$$\{R1\} = \text{new } [] \text{ mapA } f_d ++ \text{new } [] \text{ mapA } f_i ++ [F_x] \quad \{R0\} = \text{new } [] \text{ mapA } f_d ++ \text{new } [] \text{ mapA } f_i ++ [F_x]$$

$$\{R0\} = \text{new } [] \text{ mapA } f_d ++ \text{new } [] \text{ mapA } f_i ++ [F_x] = (F_x : []) \checkmark$$

$$\{++0\} [] ++ ys = ys$$

$$\{++1\} (x : xs) ++ ys = x : (xs ++ ys)$$

Lema 1 $\forall f :: a \rightarrow b . \text{map } f (xs ++ ys) = \text{map } f xs ++ \text{map } f ys$

Inducción en xs $P(xs) = \forall ys . \forall f :: a \rightarrow b . \text{map } f (xs ++ ys) = \text{map } f xs ++ \text{map } f ys$

Caso Base:

$$P([]) = \forall ys . \forall f :: a \rightarrow b . \text{map } f ([] ++ ys) = \text{map } f [] ++ \text{map } f ys$$

$$\{++0\} \text{ map } f ys \quad \{R0\} = [] ++ \text{map } f ys$$

$$\{++0\} = \text{map } f ys \checkmark$$

Caso Inductivo $\forall x :: a . P(xs) = P(x : xs) \quad H_1 \equiv P(xs) = \forall ys . \forall f :: a \rightarrow b . \text{map } f (xs ++ ys) = \text{map } f xs ++ \text{map } f ys$

QVA $P(x : xs) = \forall ys . \forall f :: a \rightarrow b . \text{map } f (x : xs ++ ys)$

$$\text{map } f (x : xs) ++ ys = \text{map } f (x : xs) + + \text{map } f ys$$

$$\{++1\} = \text{map } f (x : (xs ++ ys)) \quad \{M1\} = (F_x : \text{map } f xs) + + \text{map } f ys$$

$$\text{demo2} \quad \frac{ys : ys}{\forall ys . \forall f :: a \rightarrow b . \text{map } f (x : xs ++ ys) = \text{map } f x + + \text{map } f xs + + \text{map } f ys} \checkmark$$

Caso Base $\text{new } [] ([] ++ ys) = \text{new } [] ys ++ \text{new } [] []$

$$\{++0\} \text{ new } [] ys \quad \{R0\} = \text{new } [] ys + + [] \quad \text{preguntas dudas}$$

se logra por ordenación de ultima derivación en este caso de rebote temporal

$$= \text{new } [] ys \checkmark$$

Caso Inductivo $H_1 = \text{new } [] (xs ++ ys) = \text{new } [] ys + + \text{new } [] xs$

QVA $\text{new } [] ((x : xs ++ ys) = \text{new } [] ys + + \text{new } [] (x : xs))$

$$\{P1\} = \text{new } ([] (xs ++ ys)) = \text{new } [] ys + + \text{new } ([] xs) xs$$

$$\text{demo3} \quad \frac{ys : ys}{\text{new } [] (xs ++ ys) + + (x : xs) = \text{new } [] ys + + \text{new } [] xs + + (x : xs)} \checkmark$$

$$\{H1\} = \text{new } [] ys + + \text{new } [] xs + + (x : xs)$$

Lema 3

$\forall xs, Ac : a . \text{new } Ac xs = (\text{new } [] xs) + + Ac$

Caso Base $\text{new } Ac [] = (\text{new } [] []) + + Ac$

$$\{R0\} = Ac$$

$$\{R0\} = [] + + Ac$$

$$\{++0\} = Ac \checkmark$$

Caso Inductivo $H_1 = \text{new } Ac xs = (\text{new } [] xs) + + Ac$

QVA $\text{new } Ac (x : xs) = (\text{new } [] (x : xs)) + + Ac$

$$\{R1\} \text{ new } (x : Ac) xs$$

$$\{M1\} ((\text{new } [] xs) + + (x : Ac))$$

$$\{H1\} = (\text{new } [] xs) + + (x : []) + + Ac$$

$$\{++1\} = (\text{new } [] xs) + + x : ([] + + Ac)$$

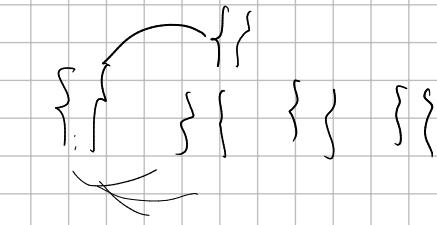
$$\{++0\} = \text{new } [] xs + + (x : Ac) \checkmark$$

Demostrar que para toda $f :: a \rightarrow b$ vale $\text{map } f . \text{post} = \text{rev } \square . \text{map } f$.

Ejercicio 2 (Programación funcional / resolución). Suponemos que contamos con los siguientes tipos de datos y funciones en Haskell para modelar la resolución SLD:

- CláusulaDeDefinición — tipo de las cláusulas definición.
- CláusulaObjetivo — tipo de las cláusulas objetivo.
- esVacia :: CláusulaObjetivo \rightarrow Bool — función que indica si una cláusula es \square .
- resolvete :: CláusulaObjetivo \rightarrow CláusulaDeDefinición \rightarrow CláusulaObjetivo — función que calcula la resolvente entre dos cláusulas.

Definir una función existeRefutaciónSLD :: [CláusulaDeDefinición] \rightarrow CláusulaObjetivo \rightarrow Bool que indica si existe una refutación SLD. En caso de que la refutación exista, la función debe devolver True. En caso de que la refutación no exista, la función debe devolver False, o posiblemente indefinirse (por no terminación).



Existe Refutación SLD :: [CláusulaDeDefinición] \rightarrow CláusulaObjetivo \rightarrow Bool

$$\text{if } [\] \text{ CO} = \text{False}$$

$$\text{if } (x : xs) \text{ CO} = \text{if EsVacia(resolvente } x \text{ CO) then True}$$

Else ExisteRefutación xs CO || ExisteRefutación (x : xs) (resolvente } x CO)

Ejercicio 4 (Deducción natural). Se define un nuevo conectivo binario \oplus , con las siguientes reglas de introducción y eliminación:

$$\frac{\Gamma, \neg\tau, \neg\sigma \vdash \perp}{\Gamma \vdash \tau \oplus \sigma} \oplus_i \quad \frac{\Gamma \vdash \tau \oplus \sigma \quad \Gamma, \tau \vdash \perp \quad \Gamma, \sigma \vdash \perp}{\Gamma \vdash \perp} \oplus_e$$

Dar derivaciones para los siguientes secuentes:

$$a) \tau \vee \sigma \vdash \tau \oplus \sigma$$

$$b) \tau \oplus \sigma \vdash \tau \vee \sigma$$

$$\frac{\overline{\alpha}}{\overline{\Gamma \vdash t \vee \sigma}} \text{ ve} \quad \frac{\overline{\alpha}}{\overline{\Gamma \vdash t \oplus \sigma}} \text{ ve} \quad \frac{\overline{\alpha} \quad \overline{\Gamma \vdash t \oplus \sigma} \quad \overline{\Gamma \vdash t \vee \sigma}}{\overline{\Gamma \vdash t \oplus \sigma \vee t \vee \sigma}} \text{ PBC}$$

$\oplus e$

$$\frac{\overline{\alpha} \quad \overline{\alpha} \quad \overline{\alpha} \quad \overline{\alpha}}{\overline{\Gamma \vdash t \vee \sigma \quad \Gamma \vdash t \oplus \sigma \quad \Gamma \vdash t \vee \sigma \quad \Gamma \vdash t \oplus \sigma}} \text{ ve}$$

$$\frac{\overline{\alpha} \quad \overline{\alpha} \quad \overline{\alpha} \quad \overline{\alpha}}{\overline{\Gamma \vdash t \rightarrow (\tau \vee \sigma) \quad \Gamma \vdash t \rightarrow (\tau \oplus \sigma)}} \text{ MT} \quad \frac{\overline{\alpha} \quad \overline{\alpha} \quad \overline{\alpha} \quad \overline{\alpha}}{\overline{\Gamma \vdash t \oplus \tau \quad \Gamma \vdash t \vee \sigma}} \text{ PBC}$$

\oplus

Ejercicio 5 (Resolución). Dar ejemplos de fórmulas que producen el siguiente comportamiento en el método de resolución.

- Hay secuencias de aplicaciones de la regla de resolución que producen infinitas cláusulas distintas. Hay una refutación que conduce a la cláusula vacía.
- Hay secuencias de aplicaciones de la regla de resolución que producen infinitas cláusulas distintas. No hay una refutación que conduce a la cláusula vacía.

No es completa.

Ejemplo

$$\{\{P(x), P(y)\}, \{\neg P(z), \neg P(w)\}\} \text{ es insatisfacible.}$$

No es posible alcanzar la cláusula vacía $\{\}$ con resolución binaria.

$$\begin{aligned} a) & \quad \{ P(a) \} \quad \{ \neg P(x), P(f(x)) \} \quad \{ \neg P(f(a)) \} \\ b) & \quad \{ P(a) \} \quad \{ \neg P(x), P(f(x)) \} \end{aligned}$$

Ejemplo — no terminación

La siguiente fórmula σ no es válida:

$$\forall x. (P(\text{succ}(x)) \Rightarrow P(x)) \Rightarrow P(0)$$

Tratemos de probar que es válida usando el método de resolución. Para ello pasamos $\neg\sigma$ a forma clausal:

$$\frac{\{\neg P(\text{succ}(x)), P(x)\}, \{\neg P(0)\}}{1 \quad 2}$$

► De 1 y 2 obtenemos 3 = $\{\neg P(\text{succ}(0))\}$.

► De 1 y 3 obtenemos 4 = $\{\neg P(\text{succ}(\text{succ}(0)))\}$.

► De 1 y 4 obtenemos 5 = $\{\neg P(\text{succ}(\text{succ}(\text{succ}(0))))\}$.

...

Ejercicio 2 (Razonamiento ecuacional). Notamos Nat al tipo de los naturales (enteros no negativos). Contamos con la operación que devuelve el máximo entre dos naturales:

max :: Nat → Nat
max x y := if x > y then x else y

Se asume ya demostrado que la operación max es commutativa, asociativa y que 0 es el elemento neutro, es decir:

- Commutatividad: $\forall x y : \text{Nat}. \text{max } x y = \text{max } y x$.

- Asociatividad: $\forall x y z : \text{Nat}. \text{max } x (\text{max } y z) = \text{max } (\text{max } x y) z$.

- Elemento neutro $\forall x : \text{Nat}. \text{max } 0 x = x$.

Se definen además dos funciones maxr y maxl para calcular el máximo elemento de una lista de números naturales:

maxr :: [Nat] → Nat	maxl :: [Nat] → Nat
{MRO} maxr [] = 0	{MLO} maxl ac [] = ac
{MRI} maxr (x : xs) = max x (maxr xs)	{MLI} maxl ac (x : xs) = maxl (max x ac) xs

La funciones (++) y reverse son las usuales:

(++) :: [a] → [a] → [a]	reverse :: [a] → [a]
{AO} [] ++ ys = ys	{RO} reverse [] = []
{A1} (x : xs) ++ ys = x : (xs ++ ys)	{R1} reverse (x : xs) = reverse xs ++ [x]

Demostar que $\forall xs : [\text{Int}]. \text{maxr}(\text{reverse } xs) = \text{maxl } 0 xs$

$$H_1 \equiv P(8) = \text{maxr}(\text{reverse } xs) = \text{maxl } 0 xs$$

QED $P(x:xs)$

$$\text{maxr}(\text{reverse}(x:xs)) = \text{maxl } 0 (x:xs)$$

$$\begin{aligned} &\{R1\} \text{maxr}(\text{reverse } xs ++ [x]) \\ &\{L1\} \text{maxr}(\text{maxr}(\text{reverse } xs)) (\text{maxr}[x]) \\ &\{H1\} \text{maxr}(\text{maxl } 0 xs) (\text{maxr}[x]) \\ &\{M1\} \text{maxr}(\text{maxl } 0 xs) \text{max } x \text{ maxr } [] \\ &\{M2\} \text{maxr}(\text{maxl } 0 xs) (\text{max } x \circ) \\ &\{EN\} \text{maxr}(\text{maxl } 0 xs) x \\ &\{\text{Comillas}\} \text{max } x (\text{maxl } 0 xs) \end{aligned}$$

$$\{M1\} = \text{maxl}(\text{max } x \circ) xs$$

$$\{EN\} = \text{maxl } x \times xs$$

$$\{L2\} = \text{max } x (\text{maxl } 0 xs)$$

$$\text{Lema 1} \quad \text{maxr}(xs ++ ys) = \text{max} \text{ maxr}(xs) \text{ maxr}(ys) \equiv H_1$$

$$\begin{aligned} \text{(Caso base)} \text{maxr}([] ++ ys) &= \text{max} \text{ maxr}([]) \text{ maxr}(ys) \\ \text{maxr}(ys) &= \text{max } 0 \text{ maxr}(ys) \\ &= \text{maxr } ys \checkmark \end{aligned}$$

Casoductivo

$$\begin{aligned} \text{maxr}(x:xs ++ ys) &= \text{max} \text{ maxr}(x:xs) \text{ maxr}(ys) \\ \{A1\} \text{maxr}(x:(xs ++ ys)) &= \text{max } x (\text{maxr } xs) \text{ maxr } ys \\ \{M2\} \text{max } x \times (\text{maxr } (xs ++ ys)) &= \text{max } x (\text{max } (\text{maxr } xs) (\text{maxr } ys)) \checkmark \\ \{H1\} \text{max } x \times (\text{maxr } xs) (\text{maxr } ys) & \end{aligned}$$

$$\text{Lema 2 } x:\text{int} \text{ maxl } x \times xs = \text{max } x (\text{maxl } 0 xs), \equiv H_1$$

$$\text{(Caso base)} \text{maxl } x \times [] = \text{max } x (\text{maxl } 0 [])$$

$$\begin{aligned} \{M1\} &= x \\ \{L1\} &= \text{max } x \times 0 \\ \{EN\} &= x \checkmark \end{aligned}$$

Casoductivo

$$\begin{aligned} y:\text{int} \text{ maxl } y (x:xs) &= \text{max } y (\text{maxl } 0 (x:xs)) \\ \{M1\} \text{maxl } (y \times) xs &= \text{max } y (\text{maxl } 0 xs) \\ \{H1\} \text{maxl } (y \times) (\text{maxl } 0 xs) &= \text{max } y (\text{maxl } x \times xs) \\ \{\text{Comillas}\} \text{maxl } (y \times x) (\text{maxl } 0 xs) &= \text{max } y (\text{max } x (\text{maxl } 0 xs)) \\ \{\text{Asociatividad}\} \text{maxl } (y \times x) (\text{maxl } 0 xs) &= \text{max } (y \times x) \text{ maxl } 0 xs \checkmark \end{aligned}$$

c) En una empresa, al final del día se realizan varias tareas rutinarias sobre distintas estructuras de datos. Estas tareas se encapsulan en bloques de código y deben ejecutarse secuencialmente sobre un mismo objeto. Se desea implementar un mecanismo genérico que permita realizar esta secuencia de operaciones sobre cualquier objeto.

Implementar en la clase Object el método runBatch, que reciba como parámetro una colección de bloques. Cada bloque debe tomar el objeto receptor como argumento, y se debe ejecutar en orden. El método debe devolver el resultado de la ejecución del último bloque.

Por ejemplo:

```
| lista resultado |
lista := OrderedCollection new.
resultado := lista runBatch: #(
```

- [:obj | obj add: 1]
- [:obj | obj add: 2]
- [:obj | 'último paso']

Al final de la ejecución, resultado es 'último paso', mientras que lista contiene 1 y 2.

object >> runBatch : Blocks
| result |

Blocks do: [:block| result := block value: self].

| result |

Por inducción estructural en Xs

$$P(xs) \equiv \text{maxr } (\text{reverse } xs) = \text{maxl } 0 xs$$

$$\text{Caso base } P([]) = \text{maxr } (\text{reverse } []) = \text{maxl } 0 []$$

$$\{R0\} = \text{maxr } [] \quad \{M1\} = 0 \checkmark$$

$$\{M2\} = 0$$

$$\text{Casoductivo: } f: x:\text{Int} \quad P(xs) \rightarrow P(x:xs)$$

$$\text{maxr}(\text{reverse}(x:xs)) = \text{maxl } 0 (x:xs)$$

$$\{R1\} \text{maxr}(\text{reverse } xs ++ [x])$$

$$\{L1\} \text{maxr}(\text{maxr}(\text{reverse } xs)) (\text{maxr}[x])$$

$$\{H1\} \text{maxr}(\text{maxl } 0 xs) (\text{maxr}[x])$$

$$\{M1\} \text{maxr}(\text{maxl } 0 xs) \text{max } x \text{ maxr } []$$

$$\{M2\} \text{maxr}(\text{maxl } 0 xs) (\text{max } x \circ)$$

$$\{EN\} \text{maxr}(\text{maxl } 0 xs) x$$

$$\{\text{Comillas}\} \text{max } x (\text{maxl } 0 xs)$$

$$\{M1\} = \text{maxl}(\text{max } x \circ) xs$$

$$\{EN\} = \text{maxl } x \times xs$$

$$\{L2\} = \text{max } x (\text{maxl } 0 xs)$$

object >> runBatch : Blocks
| result |

Blocks do: [:block| result := block value: self].

| result |

La lista se actualiza sola.

Este es poro
devuelve "ultimo paso"
se rellena.