Ejercicio 17 ★

Sea $o \stackrel{\text{def}}{=} [a = \varsigma(x)(x.a \Leftarrow \varsigma(y)(y.a \Leftarrow \varsigma(z)[]))]$. Mostrar cómo reduce $o.a.a$.

$o' = [a = \varsigma(x)(x.a \Leftarrow \varsigma(y)[])]$ $\qquad$ $o'' = [a = \varsigma(x)[]]$

$$\dfrac{\dfrac{[obj]\ \dfrac{}{0 \to 0}}{[sel]\ \dfrac{0.a \to o'}{0.a.a \to o''}}\ [sel] \qquad \dfrac{[obj]\ \dfrac{}{o' \to o'}}{o'.a \Leftarrow \varsigma(y)[] \to o''}\ [upd]}{[obj]\ \dfrac{}{0 \to 0}\ [sel]}$$

Ejercicio 18 ★

a) Definir *true* y *false* como objetos con los siguientes tres métodos: *not*, *if*, e *ifnot*. Notar que tanto *if* como *ifnot* deberán retornar una función binaria. Las operaciones deberían satisfacer las siguientes igualdades:

$true.not = false$	$true.if\ (v_1)\ (v_2) = v_1$	$false.if\ (v_1)\ (v_2) = v_2$
$false.not = true$	$true.ifnot\ (v_1)\ (v_2) = v_2$	$false.ifnot\ (v_1)\ (v_2) = v_1$

b) Definir *and* y *or* como objetos que se comporten como funciones que esperen dos argumentos (para poder escribir expresiones como *and(true)(false)*, etc.).

a) $true \stackrel{\text{def}}{=} [\ if = (\lambda x \lambda y)\ x,$

$ifnot = (\lambda x \lambda y)\ y,$

$not = \varsigma(x)[\ if = (\lambda x \lambda y)\ y,$

$ifnot = (\lambda x \lambda y)\ x,$

$not = x\ ]\ ]$

$false \stackrel{\text{def}}{=} true.not$

b) $and \stackrel{\text{def}}{=} (\lambda x \lambda y)\ x.if(y, false)$

$or \stackrel{\text{def}}{=} (\lambda x \lambda y)\ x.if(true, y)$

Ejercicio 19 ★

a) Definir el objeto *origen* que representa el origen de coordenadas en dos dimensiones. Este objeto provee tres operaciones: los observadores x e y y mv tal que $origen.mv\ (v)(w)$ desplaza a *origen* v unidades a la derecha y w unidades hacia arriba.

b) Definir una clase *Punto*, cuyas instancias proveen las operaciones x, y y mv. Considerar que los puntos se crean con sus coordenadas en 0.

c) Mostrar como reduce *Punto.new*.

d) Definir la subclase *PuntoColoreado*, que permite construir instancias de puntos que tienen asociado un color. Además del método *new*, que crea puntos blancos, la clase debe contar con un método que cree puntos de un color pasado como parámetro.

a) $origen \stackrel{\text{def}}{=} [\ x = 0,\ y = 0,\ mv = \varsigma(z)(\lambda r \lambda w)\ (z.x \Leftarrow z.x + r).y \Leftarrow z.y + w\ ]$

b) $Punto \stackrel{\text{def}}{=} [\ new = \varsigma(z)[\ x = \varsigma(s)\ z.x(s),\ y = \varsigma(s)\ z.y(s),\ mv = \varsigma(s)(\lambda v \lambda w)\ z.mv(s,v,w)\ ],$

$x = (\lambda s)\ 0,$

$y = (\lambda s)\ 0,$

$mv = (\lambda s \lambda r \lambda w)\ (s.x \Leftarrow s.x + r).y \Leftarrow s.y + w,$

$]$

c)

$0 \stackrel{\text{def}}{=} Punto.new\{Punto/z\} = [x = \varsigma(s)\ Punto.x(s),\ y = \varsigma(s)\ Punto.y(s),\ mv = \varsigma(s)(\lambda v \lambda w)\ Punto.mv(s,v,w)]$

$$\dfrac{[sel]\ \dfrac{[obj]\ \dfrac{}{Punto \to Punto}\qquad [obj]\ \dfrac{}{0 \to 0}}{Punto.new}$$

d) $PuntoColoreado = [\ new = \varsigma(z)[\ x = \varsigma(s)\ z.x(s),\ y = \varsigma(s)\ z.y(s),\ mv = \varsigma(s)\lambda(v)\lambda(w)\ z.mv(s,v,w),\ color = \varsigma(s)\ z.color(s)\ ],$

$new\ Color = \varsigma(z)\lambda(c)\ (z.new).color \Leftarrow c,$

$x = \lambda(s)\ Punto.x(s),$

$y = \lambda(s)\ Punto.y(s),$

$mv = \lambda(s)\lambda(v)\lambda(w)\ Punto.mv(s,v,w),$

$color = \lambda(s)\ \text{"BLANCO"}$

$]$

Ejercicio 21

Definir en el cálculo de objetos, el objeto `Vacio` que representa el conjunto vacío y sabe responder los siguientes mensajes:

- `hayElementos`, que devuelve `true` si el conjunto contiene al menos un elemento.
- `agregar(x)`, que devuelve el objeto que agrega `x` al conjunto.
- `sacar(x)`, que devuelve el objeto que saca `x` del conjunto.
- `pertenece(x)`, que indica si `x` pertenece al conjunto.

Ejemplos:

`Vacio.agregar(2).sacar(2).hayElementos` → `false`;

`Vacio.agregar(2).agregar(3).sacar(2).pertenece(3)` → `true`;

Se puede suponer que la operación == está definida para los elementos del conjunto.

VACIO $\stackrel{def}{=}$ [he = false ,

agregar = ς(z)λ(s)((z.he := true). pert := λ(s') if (z.pert(s'), s == s') . sacar := λ(s') (s==s') . if (z)(z.sacar(s'), agregar(s)).

sacar = ς(z) λ(s) z.

pert = λ(s) false

]

Ejercicio 23

a) Definir en Cálculo ς el objeto `emptyList`, que representa a la lista vacía.

Este objeto debe responder al mensaje `cons`, que recibe un elemento e y devuelve una nueva lista cuya cabeza es e y que, además de `cons`, debe poder responder a los mensajes `head` y `tail`.

b) Considerar la regla APP vista en clase para reducir aplicaciones de forma simplificada:

$$\frac{a \longrightarrow v' \quad v' \equiv \lambda(x)b \quad b\{x \leftarrow s\} \longrightarrow v}{a(s) \approx v} \text{[APP]}$$

Reducir `emptyList.cons(uno).cons(dos).tail`, indicando claramente las reglas utilizadas (siendo `uno` y `dos` los valores definidos en clase).

(Recordar que siempre se puede ponerle nombre a una expresión para no tener que escribirla varias veces.)

c) Sea el siguiente objeto:

```
emptyListCheck ≝ [ isEmpty = true,
                   cons = λ(x)(emptyList.cons(x)).isEmpty := false
                 ]
```

Siendo `true` y `false` los definidos en la guía.

¿Este objeto tiene el comportamiento esperado? (El comportamiento esperado es que el valor del campo `isEmpty` sea `true` si la lista es vacía y `false` en caso contrario, para cualquier lista obtenida con aplicaciones sucesivas de `cons` y `tail` a partir de `emptyListCheck`.) Justificar o indicar dónde está el problema.

EL = [cons = ς(z) λ(e)[head = e , tail = z , cons = ς(z') λ(e')(z'.head := e').tail := z']]

ς(z) λ(e)[head = e , tail = z , cons = ς(z') λ(e')(z'.head := e').tail := z']

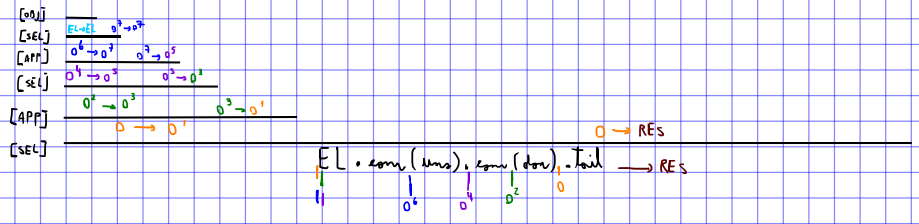
o^7 = λ(e)[head = e, tail = EL, cons = ς(z') λ(e')(z'.head := e').tail := z']

o^5 = [head = 1, tail = EL, cons = ς(z') λ(e')(z'.head := e').tail := z']

o^3 = λ(e)[head = e, tail = o^5, cons = ς(z') λ(e')(z'.head := e').tail := z']

o' = [head = 2, tail = o^5, cons = ς(z') λ(e')(z'.head := e').tail := z']

RES = o^5 = [head = 1, tail = EL, cons = ς(z') λ(e')(z'.head := e').tail := z']



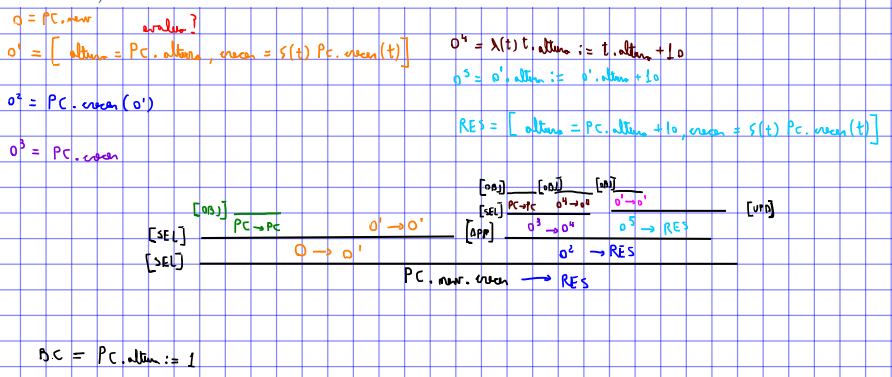
Ejercicio 22

a) Considere la siguiente clase

$$plantaClass \stackrel{\text{def}}{=} [\quad new = \varsigma(c)[altura = c.altura,\ crecer = \varsigma(t)c.crecer(t)],\\ altura = 10,\\ crecer = \varsigma(c)\lambda(t)t.altura := (t.altura + 10)]$$

b) Mostrar cómo evalua *plantaClass.new.crecer*.

c) Definir *broteClass* sobreescribiendo en *plantaClass* la altura inicial por 1. La solución debería aprovechar *plantaClass* para seguir compartiendo futuras modificaciones de *plantaClass* (por ejemplo, nuevas versiones del método *crecer*).



PC = PC.altura := 1

d) Definir *malezaClass* sobreescribiendo *crecer* en *plantaClass* de manera tal que multiplique la altura de la planta por 2.

e) Escribir la clase *frutalClass* agregando a *plantaClass* el atributo *cantFrutos* inicializado en 0 y sobreescribiendo *crecer* de manera tal que se incremente la cantidad de frutos cada vez que la planta crezca.

f) Definir una función *aFrutal* que tome una clase de planta (*plantaClass*, *broteClass*, o *malezaClass*) que retorne una nueva clase de planta **frutal** que se derive de la clase dada.

MC = PC.crecer := λ(t)(t.altura := t.altura + t.altura)

FC = [new = ς(z)[cantF = ς(s) z.cantF(s), altura = ς(s) z.altura (s), crecer = ς(s) z.crecer(s)]

cantF = λ(s) 0,

altura = λ(s) PC.altura(s),

crecer = λ(s)((PC.crecer (s)). cantF := s.cantF + 1)

]

AF = λ(t)((PC.altura := t.altura).crecer := t.crecer)

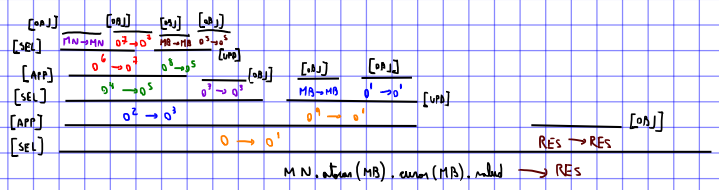
Ejercicio 2 - Cálculo Sigma

Sean los siguientes objetos que representan magos en cierto juego de fantasía:

magoBlanco $\stackrel{\text{def}}{=}$ [salud = 5, curar = $\lambda(x)x.$salud := $x.$salud.succ, estaVivo = $\varsigma(m)$ $m.$salud.mayor(0)]

magoNegro $\stackrel{\text{def}}{=}$ [salud = 3, atacar = $\lambda(x)x.$salud := $x.$salud.pred, estaVivo = $\varsigma(m)$ $m.$salud.mayor(0)]

a) Mostrar cómo reduce la expresión: magoNegro.atacar(magoBlanco).curar(magoBlanco).salud.



o = MN.altura(MB).crecer(MB)
o² = MN.altura(MB).crecer
o⁴ = MN.altura(MB)
o⁶ = MN.altura
o⁷ = λ(x) x.salud := x.salud.pred
o⁸ = MB.salud := MB.salud.pred
o⁵ = [salud = 4, crecer = ..., estaVivo = ...]
o³ = λ(x)(x.salud := x.salud).crecer)
o⁹ = MB.salud := MB.salud.pred
o' = [salud = 6, crecer = ..., estaVivo = ...]
RES = 6