1 2C2024 Parcial

1.1 Ejercicio 1

El siguiente tipo de datos sirve para representar un *buffer con historia* que permite escribir o leer en cualquiera de sus posiciones (las posiciones tienen tipo `Int`). El tipo del buffer es paramétrico en el tipo de los contenidos que se pueden guardar en él. Si se escribe dos veces en la misma posición, el nuevo contenido pisa al anterior (por simplicidad). La lectura elimina el contenido leído.

```
data Buffer a = Empty | Write Int a (Buffer a) | Read Int (Buffer a)
```

Definimos el siguiente buffer para los ejemplos:

```
buf = Write 1 'a' $ Write 2 'b' $ Write 1 'c' $ Empty
```

a) Nos piden definir **foldBuffer** y **recBuffer**

```
foldBuffer :: b -> (Int-> a -> b -> b) -> (Int-> b -> b) -> Buffer a -> b
foldBuffer fEmpty fWrite fRead buf = case buf of
    Empty -> fEmpty
    Write n s b -> fWrite n s (rec b)
    Read n b -> fRead n (rec b)
    where rec = foldBuffer fEmpty fWrite fRead

recBuffer :: b -> (Int-> a -> Buffer a -> b -> b) -> (Int-> Buffer a -> b -> b) -> Buffer a -> b
recBuffer fEmpty fWrite fRead buf = case buf of
    Empty -> fEmpty
    Write n s b -> fWrite n s b (rec b)
    Read n b -> fRead n b (rec b)
    where rec = recBuffer fEmpty fWrite fRead
```

b) Nos piden definir una función que devuelva una lista con las **posiciones ocupadas** del buffer sin repetir

```
posicionesOcupadas :: Buffer a -> [Int]
posicionesOcupadas = foldBuffer [] (\n s rec -> if (elem n rec) then rec else rec ++ [n])
                     (\n rec -> rec)
```

c) Necesitamos definir la función que devuelve el **contenido** de una posición del buffer

```
contenido :: Int -> Buffer a -> Maybe  a
contenido n = recBuffer Nothing (\m s l rec -> if n == m then Just s else Nothing)
            (\m l rec -> Nothing)
```

d) La función **completarLecturas** indica si se puede leer exitosamente en todas las lecturas del buffer

```
puedeCompletarLecturas :: (Eq a) => Buffer a -> Bool
puedeCompletarLecturas = recBuffer False (\m s l rec -> rec )
                         (\m l rec -> if (contenido m l) /= Nothing then True else False )
```

e) La función **deshacer** nos permite deshacer las últimas n-operaciones en el buffer

```
deshacer :: Buffer a -> Int -> Buffer a
deshacer = recBuffer (const Empty) (\m s l rec -> \n-> if n == 0 then Write m s l else rec (n-1)) (\m l rec -> \n -> if n == 0 then Read m l else rec (n-1))

deshacerCheck = contenido 1 (deshacer buf 2) ~> Just 'c'
```

La idea de usar currificación es para que podemos usar el número n para iterar. Usamos recBuffer para tener referencia a toda la estructura

1.2 Ejercicio 2

Considerar las siguientes definiciones

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

    const :: a -> b -> a
{C} const = (\x -> \y -> x )

    altura :: AB a -> Int
{A0}altura Nil = 0
{A1}altura (Bin i r d) = 1 + max (altura i) (altura d)
    zipAB :: AB a -> AB b -> AB (a,b)
{Z0}zipAB Nil = const Nil
{Z1}zipAB (Bin i r d) = (\t -> case t of Nil -> Nil Bin i' r' d' -> Bin (zipAB i i') (r r') (zipAB d d'))
```

a) Nos piden demostrar

$$\forall t, u : AB\ a. \quad altura\ t \geq altura\ (zipAB\ t\ u)$$

Contamos con este **lema** ya demostrado

$$\{LEMA\}\ \forall t : AB\ a.\ altura\ t \geq 0$$

Vamos a demostrarlo por inducción sobre **AB a**. Enunciamos

$$\forall x : AB. \quad P(x) = altura\ x \geq altura\ (zipAB\ x\ u)$$

- Caso base :

$$P(Nil) = altura\ Nil \geq altura\ (zipAB\ Nil\ u)$$
$$\overset{\{A0\}}{\equiv} 0 \geq altura\ (zipAB\ Nil\ u)$$
$$\overset{\{Z0\}}{\equiv} 0 \geq altura\ Const\ Nil$$
$$\overset{\{C\}}{\equiv} 0 \geq altura\ Nil$$
$$\overset{\{A0\}}{\equiv} 0 \geq 0$$
$$\equiv True$$

- Ahora podemos enunciar $\forall i, d : AB\ a, r : a. \quad P(i) \wedge P(d) \implies P(Bin\ i\ r\ d)$. Asumimos verdaderas $P(i) \wedge P(d)$ y serán nuestras **HI**. Con esto **QVQ**

$$P(Bin\ i\ r\ d) = altura\ (Bin\ i\ r\ d) \geq altura\ (zipAB\ (Bin\ i\ r\ d)\ u)$$
$$\overset{\{A1\}}{\equiv} 1 + \max(altura\ i)(altura\ d) \geq altura\ (zipAB\ (Bin\ i\ r\ d)\ u)$$
$$\overset{\{Z1\}}{\equiv} \ldots \geq altura\ (\backslash t\ case\ t\ of Nil \rightarrow Nil\ \ Bin\ i'\ r'\ d' \rightarrow (zipAB\ i\ i')\ (r\ r')\ (zipAB\ d\ d'))\ u)$$
$$\overset{\beta}{\equiv} \ldots \geq altura\ (case\ u\ of Nil \rightarrow Nil\ \ Bin\ ui\ ur\ ud \rightarrow (zipAB\ i\ ui)\ (r\ ur)\ (zipAB\ d\ ud))$$

De esto se desprenden dos casos, que los resolvemos por extensionalidad

  * Caso **u es Nil**

$$\ldots \geq altura\ (case\ Nil\ of Nil \rightarrow Nil\ \ Bin\ ui\ ur\ ud \rightarrow (zipAB\ i\ ui)\ (r\ ur)\ (zipAB\ d\ ud))$$
$$\overset{case\_Nil}{\equiv} \ldots \geq altura\ Nil$$
$$\overset{\{A0\}}{\equiv} \ldots \geq 0$$
$$\equiv 1 + \max(altura\ i)(altura\ d) \geq 0$$
$$\overset{\{LEMA\}}{\equiv} True$$

  * Caso **u no es Nil**

$$\ldots \geq altura\ (case\ Bin\ ui\ ur\ ud\ of Nil \rightarrow Nil\ \ Bin\ ui\ ur\ ud \rightarrow (zipAB\ i\ ui)\ (r\ ur)\ (z$$
$$\overset{case\_Bin}{\equiv} \ldots \geq altura((zipAB\ i\ ui)\ (r\ ur)\ (zipAB\ d\ ud))$$
$$\equiv 1 + \max(altura\ i)(altura\ d) \geq altura((zipAB\ i\ ui)\ (r\ ur)\ (zipAB\ d\ ud))$$
$$\overset{\{A1\}}{\equiv} 1 + \max(altura\ i)(altura\ d) \geq 1 + \max(altura(zipAB\ i\ ui))\ (altura(zipAB\ d\ ud))$$

Recordemos las HI :

$$P(i) = altura\ i \geq altura(zipAB\ i\ u)$$
$$P(d) = altura\ d \geq altura(zipAB\ d\ u)$$

$$\overset{\{HI\}}{\equiv} 1 \geq 1$$
$$\equiv True$$

b) Demostrar sin principios clásicos :

$$((\rho \wedge \sigma) \vee (\rho \wedge \tau)) \implies (\sigma \wedge \rho) \vee \tau$$

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\Gamma' \vdash (\rho \wedge \sigma) \vee (\rho \wedge \tau)$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\Gamma', (\rho \wedge \sigma) \vdash \rho \wedge \sigma$}
\RightLabel{$\wedge_{e_2}$}
\UnaryInfC{$\Gamma', (\rho \wedge \sigma) \vdash \sigma$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\Gamma', (\rho \wedge \sigma) \vdash \rho \wedge \sigma$}
\RightLabel{$\wedge_{e_1}$}
\UnaryInfC{$\Gamma', (\rho \wedge \sigma) \vdash \rho$}
\RightLabel{$\wedge_i$}
\BinaryInfC{$\Gamma', (\rho \wedge \sigma) \vdash \sigma \wedge \rho$}
\RightLabel{$\vee_{i_1}$}
\UnaryInfC{$\Gamma', (\rho \wedge \sigma) \vdash (\sigma \wedge \rho) \vee \tau$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\Gamma', (\rho \wedge \tau) \vdash \rho \wedge \tau$}
\RightLabel{$\wedge_{e_2}$}
\UnaryInfC{$\Gamma', (\rho \wedge \tau) \vdash \tau$}
\RightLabel{$\vee_{i_2}$}
\UnaryInfC{$\Gamma', (\rho \wedge \tau) \vdash (\sigma \wedge \rho) \vee \tau$}
\RightLabel{$\vee_e$}
\TrinaryInfC{$\Gamma, ((\rho \wedge \sigma) \vee (\rho \wedge \tau)) \vdash (\sigma \wedge \rho) \vee \tau$}
\RightLabel{$\Rightarrow_i$}
\UnaryInfC{$\Gamma \vdash ((\rho \wedge \sigma) \vee (\rho \wedge \tau)) \implies (\sigma \wedge \rho) \vee \tau$}
\end{prooftree}
$$

1.3 Ejercicio 3

Se desea extender el cálculo lambda simplemente tipado para modelar **Árboles ternarios**. Por eso, se extienden los tipos y expresiones como sigue:

$$\tau ::= \ldots \mid AT(\tau)$$
$$M ::= \ldots \mid TNil_\tau \mid Tern(M, M, M, M) \mid foldAT\ M\ TNil \rightsquigarrow M; Tern(x, ri, rm, rd) \rightsquigarrow M$$

a) Nos piden introducir las reglas de tipo para esta extensión

$$\frac{}{\Gamma \vdash TNil_\tau : AT(\tau)}\ \text{T-TNIL}$$

$$\frac{\Gamma \vdash A : \tau \qquad \Gamma \vdash B : AT(\tau) \qquad \Gamma \vdash C : AT(\tau) \qquad \Gamma \vdash D : AT(\tau)}{\Gamma \vdash Tern(A, B, C, D) : AT(\tau)}\ \text{T-TERN}$$

$$\frac{\Gamma \vdash A : AT(\tau) \qquad \Gamma \vdash B : \rho \qquad \Gamma, x : \tau, ri : \rho, rm : \rho, rd : \rho \vdash C : \rho}{\Gamma \vdash foldAT\ A\ TNil \rightsquigarrow B; Tern(x, ri, rm, rd) \rightsquigarrow C : \rho}\ \text{T-FOLD}$$

b) Nos piden dar el conjunto de valores extendido

$$V ::= \ldots \mid TNil_\tau \mid Tern(V, V, V, V)$$

las reglas de cómputo...

$$\frac{}{foldAT\ TNil_\tau\ TNil \rightsquigarrow B; Tern(x, ri, rm, rd) \rightsquigarrow C \to B}\ \text{E-FOLDNIL}$$

$$\frac{}{foldAT\ Tern(V_1, V_2, V_3, V_4)\ TNil \rightsquigarrow B; Tern(x, ri, rm, rd) \rightsquigarrow C}\ \text{E-FOLDVALUES}$$

$\to C\{x := V_1\}\{ri := foldAT\ V_2 \ldots\}\{rm := foldAT\ V_3 \ldots\}\{rd := foldAT\ V_4 \ldots\}$ Las de congruencias son todas aquellas en las que $M \to M'$ o sea :
Para el Tern...

- $Tern(M, V_2, V_3, V_4) \to Tern(M', V_2, V_3, V_4)$
- $Tern(O, M, V_3, V_4) \to Tern(O, M', V_3, V_4)$
- $Tern(O, P, M, V_4) \to Tern(O, P, M', V_4)$
- $Tern(O, P, Q, M) \to Tern(O, P, Q, M')$

Para el foldAT...

- $foldAT\ M\ TNil \rightsquigarrow B; Tern(x, ri, rm, rd) \rightsquigarrow C \to foldAT\ M'\ TNil \rightsquigarrow B; Tern(x, ri, rm, rd) \rightsquigarrow C$

- Reducir :

  $(\lambda t : AT(Nat).\ foldAT\ TNil \rightsquigarrow False;\ Tern(x, ri, rm, rd) \rightsquigarrow isZero(Pred(x)))\ \ Tern(\underline{1}, TNil_{Nat}, TNil_{Nat}, TNil_{Nat})$

  $\stackrel{\beta}{\to} foldAT\ Tern(\underline{1}, TNil_{Nat}, TNil_{Nat}, TNil_{Nat})\ TNil \rightsquigarrow False;\ Tern(x, ri, rm, rd) \rightsquigarrow isZero(Pred(x)))$

  $\stackrel{E-FOLDVALUES}{\to} isZero(Pred(\underline{1}))$

  $\equiv isZero(Pred(Succ(zero)))$

  $\stackrel{E-PREDSUCC}{\to} isZero(zero)$

  $\stackrel{E-ISZEROZERO}{\to} True$