PLP - Clase de repaso

Para el Primer Parcial

Departamento de Computación
FCEyN UBA

1c2024

Programación Funcional - 1er Recuperatorio 1c2023

Consideremos la siguiente representación para naves espaciales, donde dos naves pueden fusionarse a través de un módulo conector para formar una nave más grande y potente:

```haskell
data Componente
    = Contenedor
    | Motor
    | Escudo
    | Cañón
    deriving Eq

data NaveEspacial
    = Módulo Componente NaveEspacial NaveEspacial
    | Base Componente
    deriving Eq
```

Vamos a querer definir una serie de funciones.

Inciso a

`recNave` y `foldNave`:
Los esquemas de recursión primitiva y estructural respectivamente. Indicar los tipos de ambos esquemas. Para la defnción de recNave se puede utilizar recursión explicita. En cambio foldNave deberá definirse a partir de recNave.

```
recNave :: ?

foldNave :: ?
```

Inciso a

```haskell
recNave :: (Componente -> NaveEspacial -> NaveEspacial
                                       -> a -> a -> a)
        -> (Componente -> a) -> NaveEspacial -> a

recNave f1 f2 n = case n of
    Módulo c n1 n2 -> f1 c n1 n2 (rec n1) (rec n2)
    Base c -> f2 c
  where rec = recNave f1 f2

foldNave :: (Componente -> a -> a -> a)
         -> (Componente -> a) -> NaveEspacial -> a

foldNave f1 f2 = recNave (\c _ _ -> f1 c) f2
```

Inciso b

`espejo :: NaveEspacial -> NaveEspacial`:
Dada una nave devuelve otra con los mismos componentes, pero en posiciones espejadas (los de la derecha a la izquierda y viceversa)

Por ejemplo:

```
> espejo (Módulo Motor (Base Escudo) (Base Contenedor))
Módulo Motor (Base Contenedor) (Base Escudo)
```

Inciso b

`espejo :: NaveEspacial -> NaveEspacial`:
Dada una nave devuelve otra con los mismos componentes, pero en posiciones espejadas (los de la derecha a la izquierda y viceversa)

```
espejo :: NaveEspacial -> NaveEspacial
espejo = foldNave (\c r1 r2 -> Módulo c r2 r1) Base
```

Inciso c

`esSubnavePropia :: NaveEspacial -> NaveEspacial -> Bool`:
Dadas dos naves, indica si la primera está contenida propiamente dentro de la segunda.
Solo se considerarán subnaves propias aquellas que se encuentren al fondo. Es decir, que se encuentren en la segunda nave como "subárboles" sin otros componentes más abajo, y que no sean iguales a la segunda nave.

Por ejemplo:

```
> esSubnavePropia (Base Escudo) (Módulo Motor (Base Escudo) (Base Motor))
True
```

Inciso c

`esSubnavePropia :: NaveEspacial -> NaveEspacial -> Bool`:
Dadas dos naves, indica si la primera está contenida propiamente dentro de la segunda.
Solo se considerarán subnaves propias aquellas que se encuentren al fondo. Es decir, que se encuentren en la segunda nave como "subárboles" sin otros componentes más abajo, y que no sean iguales a la segunda nave.

```
esSubnavePropia :: NaveEspacial -> NaveEspacial -> Bool
esSubnavePropia n1 =
  recNave (\_ sn1 sn2 r1 r2 -> sn1 == n1 ||
                               sn2 == n1 || r1 || r2)
          (const False)
```

Inciso d

`truncar :: NaveEspacial -> Integer -> NaveEspacial`:
Dada una nave y un número natural n, devuelve una nave con los niveles 0 a n de la original, siendo el nivel 0 la raíz.

Inciso d

`truncar :: NaveEspacial -> Integer -> NaveEspacial`:
Dada una nave y un número natural n, devuelve una nave con los niveles 0 a n de la original, siendo el nivel 0 la raíz.

```haskell
truncar :: NaveEspacial -> Integer -> NaveEspacial
truncar =
    foldNave (\c r1 r2 -> \i ->
                    if i == 0
                    then (Base c)
                    else (Módulo c (r1 (i-1)) (r2 (i-1))))
             (\c -> \i -> Base c)
```

Dada la siguiente función[1]:

```
truncar ::  AB a -> Int -> AB a
truncar Nil _ = Nil
truncar (Bin i r d) n = if n == 0 then Nil
                else Bin (truncar i (n-1)) r (truncar d (n-1))
```

Y los siguientes lemas:

1. $\forall$ `x::Int` . $\forall$ `y::Int` . $\forall$ `z::Int` .
   max (min x y) (min x z) = min x (max y z)
2. $\forall$ `x::Int` . $\forall$ `y::Int` . $\forall$ `z::Int` . z + min x y = min (z+x) (z+y)

Demostrar las siguientes propiedades:

1. $\forall$ `t::AB a` . altura t $\geq$ 0
2. $\forall$ `t::AB a` . $\forall$ `n::Int` . `n` $\geq$ `0` $\Rightarrow$
   `(altura (truncar t n)` = `min n (altura t))`

[1] También podría escribirse usando foldAB.

Definiciones y lemas

```
truncar ::  AB a -> Int -> AB a
{T0} truncar Nil _ = Nil
{T1} truncar (Bin i r d) n = if n == 0 then Nil
           else Bin (truncar i (n-1)) r (truncar d (n-1))

foldAB ::  b -> (b -> a -> b -> b) -> AB a -> b
{F0} foldAB cNil cBin Nil = cNil
{F1} foldAB cNil cBin (Bin i r d) = cBin (rec i) r (rec d)
                   where rec = foldAB cNil cBin

altura ::  AB a -> Int
{A0} altura = foldAB 0 (\ri x rd -> 1 + max ri rd)
```

1. ∀ x::Int . ∀ y::Int . ∀ z::Int .
   max (min x y) (min x z) = min x (max y z)
2. ∀ x::Int . ∀ y::Int . ∀ z::Int . z + min x y = min (z+x) (z+y)

Deducción natural: práctica 3, ejercicio 6)*VIII*

Demostrar en deducción natural que la siguiente fórmula es un teorema.
Es necesario usar principios de razonamiento clásicos para la dirección $\Rightarrow$.
No se permite usar principios de razonamiento clásicos para la dirección $\Leftarrow$.

$$\neg(\rho \wedge \sigma) \Leftrightarrow (\neg\rho \vee \neg\sigma)$$

Reglas de deducción natural

$$\frac{}{\Gamma, \tau \vdash \tau}\ \text{ax}$$

$$\frac{\Gamma \vdash \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau \wedge \sigma}\ \wedge_i \qquad \frac{\Gamma \vdash \tau \wedge \sigma}{\Gamma \vdash \tau}\ \wedge_{e_1} \qquad \frac{\Gamma \vdash \tau \wedge \sigma}{\Gamma \vdash \sigma}\ \wedge_{e_2}$$

$$\frac{\Gamma, \tau \vdash \sigma}{\Gamma \vdash \tau \Rightarrow \sigma}\ \Rightarrow_i \qquad \frac{\Gamma \vdash \tau \Rightarrow \sigma \quad \Gamma \vdash \tau}{\Gamma \vdash \sigma}\ \Rightarrow_e$$

$$\frac{\Gamma \vdash \tau}{\Gamma \vdash \tau \vee \sigma}\ \vee_{i_1} \qquad \frac{\Gamma \vdash \sigma}{\Gamma \vdash \tau \vee \sigma}\ \vee_{i_2} \qquad \frac{\Gamma \vdash \tau \vee \sigma \quad \Gamma, \tau \vdash \rho \quad \Gamma, \sigma \vdash \rho}{\Gamma \vdash \rho}\ \vee_e$$

$$\frac{\Gamma, \tau \vdash \bot}{\Gamma \vdash \neg\tau}\ \neg_i \qquad \frac{\Gamma \vdash \tau \quad \Gamma \vdash \neg\tau}{\Gamma \vdash \bot}\ \neg_e$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \tau}\ \bot_e$$

Lógica intuicionista

$$\frac{\Gamma \vdash \neg\neg\tau}{\Gamma \vdash \tau}\ \neg\neg_e \qquad \frac{\Gamma, \neg\tau \vdash \bot}{\Gamma \vdash \tau}\ \text{PBC} \qquad \frac{}{\Gamma \vdash \tau \vee \neg\tau}\ \text{LEM}$$

Lógica clásica (las tres son equivalentes entre sí)

Se desea extender el Cálculo Lambda tipado con colas bidireccionales (también conocidas como *deque*).
Se extenderán los tipos y términos de la siguiente manera:

$$\tau ::= \cdots \mid \text{Cola}_\tau$$

$$\begin{aligned} M ::= \cdots &\mid \langle\rangle_\tau \mid M \bullet M \mid \text{próximo}(M) \mid \text{desencolar}(M) \\ &\mid \text{case } M \text{ of } \langle\rangle \rightsquigarrow M; c \bullet x \rightsquigarrow M \end{aligned}$$

donde $\langle\rangle_\sigma$ es la cola vacía en la que se pueden encolar elementos de tipo σ; $M_1 \bullet M_2$ representa el agregado del elemento M_2 al **final** de la cola M_1; los observadores próximo(M_1) y desencolar(M_1) devuelven, respectivamente, el primer elemento de la cola (el primero que se encoló), y la cola sin el primer elemento (estos dos últimos solo tienen sentido si la cola no es vacía); y el observador case M_1 of $\langle\rangle \rightsquigarrow M_2; c \bullet x \rightsquigarrow M_3$ permite operar con la cola en sentido contrario, accediendo al último elemento encolado (cuyo valor se ligará a la variable x en M_3) y al resto de la cola (que se ligará a la variable c en el mismo subtérmino).

Cálculo Lambda - 1er Parcial 1c2023

$$\begin{aligned}
\tau &::= \cdots \mid \mathsf{Cola}_\tau \\
M &::= \cdots \mid \langle\rangle_\tau \mid M \bullet M \mid \mathsf{próximo}(M) \mid \mathsf{desencolar}(M) \\
&\quad\ \mid \mathsf{case}\ M\ \mathsf{of}\ \langle\rangle \rightsquigarrow M;\, c \bullet x \rightsquigarrow M
\end{aligned}$$

1. Introducir las reglas de tipado para la extensión propuesta.

Cálculo Lambda - 1er Parcial 1c2023

$$\tau ::= \cdots \mid \text{Cola}_\tau$$
$$M ::= \cdots \mid \langle\rangle_\tau \mid M \bullet M \mid \text{próximo}(M) \mid \text{desencolar}(M)$$
$$\mid \text{case } M \text{ of } \langle\rangle \rightsquigarrow M; c \bullet x \rightsquigarrow M$$

1. Introducir las reglas de tipado para la extensión propuesta.
2. Definir el conjunto de valores y las nuevas reglas de reducción. Pueden usar los conectivos booleanos de la guía. **No es necesario escribir las reglas de congruencia,** basta con indicar cuántas son.

Cálculo Lambda - 1er Parcial 1c2023

$$\tau ::= \cdots \mid \mathsf{Cola}_\tau$$
$$M ::= \cdots \mid \langle\rangle_\tau \mid M \bullet M \mid \mathsf{próximo}(M) \mid \mathsf{desencolar}(M)$$
$$\mid \mathsf{case}\ M\ \mathsf{of}\ \langle\rangle \rightsquigarrow M; c \bullet x \rightsquigarrow M$$

1. Introducir las reglas de tipado para la extensión propuesta.
2. Definir el conjunto de valores y las nuevas reglas de reducción. Pueden usar los conectivos booleanos de la guía. **No es necesario escribir las reglas de congruencia,** basta con indicar cuántas son.
   Pista: puede ser necesario mirar más de un nivel de un término para saber a qué reduce.

Cálculo Lambda - 1er Parcial 1c2023

$$\tau ::= \cdots \mid \mathsf{Cola}_\tau$$
$$M ::= \cdots \mid \langle\rangle_\tau \mid M \bullet M \mid \mathsf{próximo}(M) \mid \mathsf{desencolar}(M)$$
$$\mid \mathsf{case}\ M\ \mathsf{of}\ \langle\rangle \rightsquigarrow M;\ c \bullet x \rightsquigarrow M$$

1. Introducir las reglas de tipado para la extensión propuesta.
2. Definir el conjunto de valores y las nuevas reglas de reducción. Pueden usar los conectivos booleanos de la guía. **No es necesario escribir las reglas de congruencia,** basta con indicar cuántas son.
   Pista: puede ser necesario mirar más de un nivel de un término para saber a qué reduce.
3. Mostrar paso por paso cómo reduce la expresión:
   $\mathsf{case}\ \langle\rangle_{\mathsf{Nat}} \bullet \underline{1} \bullet 0\ \mathsf{of}\ \langle\rangle \rightsquigarrow \mathsf{próximo}(\langle\rangle_{\mathsf{Bool}});\ c \bullet x \rightsquigarrow \mathsf{isZero}(x)$

Cálculo Lambda - 1er Parcial 1c2023

$\tau ::= \cdots \mid \mathsf{Cola}_\tau$
$M ::= \cdots \mid \langle\rangle_\tau \mid M \bullet M \mid \mathsf{próximo}(M) \mid \mathsf{desencolar}(M)$
$\quad\quad \mid \mathsf{case}\ M\ \mathsf{of}\ \langle\rangle \rightsquigarrow M; c \bullet x \rightsquigarrow M$

1. Introducir las reglas de tipado para la extensión propuesta.
2. Definir el conjunto de valores y las nuevas reglas de reducción. Pueden usar los conectivos booleanos de la guía. **No es necesario escribir las reglas de congruencia,** basta con indicar cuántas son.
   Pista: puede ser necesario mirar más de un nivel de un término para saber a qué reduce.
3. Mostrar paso por paso cómo reduce la expresión:
   $\mathsf{case}\ \langle\rangle_{\mathsf{Nat}} \bullet \underline{1} \bullet 0\ \mathsf{of}\ \langle\rangle \rightsquigarrow \mathsf{próximo}(\langle\rangle_{\mathsf{Bool}}); c \bullet x \rightsquigarrow \mathsf{isZero}(x)$
4. Definir como macro la función $\texttt{último}_\tau$, que dada una cola devuelve el último elemento que se encoló en ella. Si la cola es vacía, puede colgarse o llegar a una forma normal bien tipada que no sea un valor. Dar un juicio de tipado válido para esta función (no es necesario demostrarlo).

Utilizando el árbol de inferencia, inferir el tipo de la siguiente expresión o demostrar que no es tipable. En cada paso donde se realice una unificación, mostrar el conjunto de ecuaciones a unificar y la sustitución obtenida como resultado de la misma.

$$(\lambda f.\ \lambda x.\ f\ (f\ (\mathsf{isZero}(x))))\ x$$

Extender los intérpretes CBN y CBV para tipos suma. Luego, evaluar la siguiente expresión:

$$\mathsf{case}\,\mathsf{pred}(\underline{2})\,\mathsf{of}\ \mathsf{left}(x) \rightsquigarrow \mathsf{isZero}(x) \,||\, \mathsf{right}(y) \rightsquigarrow \mathsf{True}$$

¿Preguntas?

¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿? ? ? ? ? ? ? ? ?