Lenguajes de programación 2016-2
Práctica 4: Paradigma imperativo II: Términos anónimos y pilas de control

Noé Salomón Hernández Sánchez
Albert M. Orozco Camacho
C. Moisés Vázquez Reyes

Facultad de Ciencias UNAM

1. Índices de De Brujin

Recordemos que las expresiones del cálculo lambda sin tipos son:

$$e ::= x \mid e\ e \mid \lambda x.e$$

la idea es transformar expresiones del cálculo lambda a *términos anónimos*:

$$a ::= n \mid a\ a \mid \lambda.a$$

donde $n \in \mathbb{N}$. Las respectivas definiciones en Haskell son:

```
data E = VarE String | AppE E E | LamE String E deriving Show
```

```
data A = VarA Int | AppA A A | LamA A deriving Show
```

La explicación de las funciones que se van a implementar se darán en el laboratorio.

- `qn::CtxCan->E->A`

  Función que transforma una expresión a un término anónimo.

  ```
  > qn ["x","y"] $ AppE (AppE (VarE "z") (VarE "x")) (LamE "y" $ AppE (AppE (VarE "z") (VarE "x")) (VarE "y"))
  ```
  (1 0) (λ.(2 1) 0)

- `pn::CtxNom->A->E`

  Función que transforma un término anónimo en una expresión lambda.

  ```
  >pn ["x","y"] $ LamA $ AppA (AppA (VarA 0) (VarA 1)) (LamA $ AppA (AppA (VarA 1) (VarA 2)) (VarA 0))
  ```
  λz.(z x) (λu.(z x) u)

- `shift::Int->Int->A->A`

  Función que desplaza índices.

  ```
  >shift 1 1 (LamA $ AppA (VarA 0) (VarA 2)
  ```
  λ.0 3

- `sust::A->Int->A->A`

  Aplica una sustitución a un término anónimo.

  `>sust (LamA $ AppA (AppA (VarA 0) (VarA 2)) (VarA 1)) 1 (LamA $ AppA (VarA 0) (VarA 2))`
  $\lambda.(0\ (\lambda.0\ 3))\ 1$

- `br::A->A->A`

  Realiza la β-reducción en términos anónimos.

  `>br (LamA $ AppA (AppA (VarA 1) (VarA 0)) (VarA 2)) (LamA $ VarA 0)`
  $(0\ (\lambda.0))\ 1$

2. Maquina $\mathcal{K}$

Considera la gramática:

$$e := x \mid n \mid \mathsf{true} \mid \mathsf{false} \mid e + e \mid e * e \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid \mathsf{let}\ x = e\ \mathsf{in}\ e \mid$$

$$e < e \mid e = e \mid \neg e \mid \lambda.e \mid e\ e \mid \mathsf{fix}\ \Rightarrow e \mid \mathsf{fail} \mid \mathsf{catch}\ e\ \mathsf{ow}\ e$$

donde las variables son enteros, y las abstracciones y fix son términos anónimos; la respectiva representación en Haskell es:

```
data LamAB = Var Int |
             VNum Int   |
             VBool Bool |
             Suma LamAB LamAB |
             Prod LamAB LamAB |
             Ifte LamAB LamAB LamAB |
             Let Int LamAB LamAB |
             Menor LamAB LamAB |
             Eq LamAB LamAB |
             Neg LamAB |
             Lambda LamAB |
             App LamAB LamAB |
             Fix LamAB |
             Fail |
             CatchOw LamAB LamAB deriving (Show,Eq)
```

Para dar la semántica estática y dinámica de la máquina $\mathcal{K}$ definimos la pila de control como una lista de marcos. La implementación de los marcos se muestra más adelante. Para codificar en Haskell la pila de control tenemos:

```
type Pila = [Marco]
```

y los estados como:

```
data EstadoMK = Ev (Pila,LamAB)
              | Dv (Pila,LamAB)
              | Pg (Pila,LamAB)
```

que corresponden a evaluar, devolver un valor y propagar un error, respectivamente.

- Termina de completar la categoría de marcos, en el archivo de la práctica sólo están implementados los marcos:

```
data Marco = MSumI () LamAB  |
             MSumD LamAB ()  |
             MProdI () LamAB |
             MProdD LamAB ()  deriving (Show,Eq)
```

  que son los marcos correpondientes a la suma y el producto.

- `esFinal :: EstadoMK->Bool`

  Nos dice si un estado es final.

  - `esFinal $ Ev ([MSumI () $ VNum 4], VNum 4)`
    False
  - `esFinal $ Dv ([], VNum 4)`
    True

- `eval1 :: EstadoMK->EstadoMK`

  Realiza un paso de evaluación en la máquina $\mathcal{K}$.

  `>eval1 $ Ev ([], Suma (Suma (VNum 1) (VNum 2)) (VNum 3))`
  Ev ([MSumI () (VNum 3)],Suma (VNum 1) (VNum 2))

- `evalK :: EstadoMK->EstadoMK`

  Realiza una ejecución completa en la máquina $\mathcal{K}$.

  `>evalK $ Ev ([], Suma (Suma (VNum 1) (VNum 2)) (VNum 3))`
  Dv ([],VNum 6)

- `vt :: Ctx->LamAB->Tipo`

  Determina el tipo de la expresión `LamAB` bajo el contexto `Ctx` siguiendo las reglas de tipado vistas en el laboratorio.

  `>vt [(2,TInt)] $ Suma (Var 2) (VNum 3)`
  TInt

Para realizar la aplicación y la β-reducción deben utilizar las funciones implementadas en la sección de índices de De Brujin.