

Lenguajes de programación 2016-2

Práctica 3: Paradigma imperativo

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

Facultad de Ciencias UNAM

Dada la gramática:

$x \mid n \mid \text{true} \mid \text{false} \mid e + e \mid \text{if } e \text{ then } e \text{ else } e \mid \text{iszero } e \mid \text{let } x = e \text{ in } e \text{ end} \mid e < e \mid e = e \mid \neg e$
con $n \in \mathbb{Z}$, la extendemos con las expresiones:

$e_1 := e_2 \mid \text{ref } e \mid !e \mid e_1; e_2 \mid \text{while } e_1 \text{ do } e_2 \mid \text{unit}$

que denotan asignación, alojamiento, obtención, secuenciación, ciclos y el valor unit, respectivamente.

En esta práctica el objetivo es aumentar la funcionalidad de la práctica 1 implementando características de la programación imperativa.

La respectiva extensión en HASKELL es la siguiente:

```
data EAB = Var String
         | VNum Int
         | VBool Bool
         | Suma EAB EAB
         | Ifte EAB EAB EAB
         | Iszero EAB
         | Let String EAB EAB
         | Menor EAB EAB
         | Eq EAB EAB
         | Neg EAB
         | Asig EAB EAB
         | Ref EAB
         | Deref EAB
         | L Int
         | Seq EAB EAB
         | While EAB EAB
         | Unit deriving (Show, Eq)
```

con L el constructor para las direcciones de memoria, la cual representaremos con el tipo:

`type Mem = [(LDir,Val)]`

donde $LDir$ es un alias del tipo EAB que servirá para identificar expresiones de la forma $L \ n$. Esto es para hacer énfasis en la idea de que una dirección la representamos con el símbolo ℓ . Val también es un alias del tipo EAB que se restringe a los constructores que representan valores, esto es para enfatizar que una memoria guarda tales valores. Por lo que una memoria tiene la forma $[(L \ n_1, v_1), \dots, (L \ n_k, v_k)]$.

1. Semántica dinámica

A partir de las reglas de evaluación dadas en la Nota 3, implemente en HASKELL las siguientes funciones que requieren del uso de memoria.

■ `accessMem::LDir ->Mem->Maybe Val`

Función que recibe una etiqueta de la forma $(L\ n)$, y devuelve el valor alojado en memoria en la posición n .

- `>accessMem (L 1) [(L 0,VBool False),(L 1,VNum 3)]`
`Just (VNum 3)`
- `>accessMem (L 2) [(L 0,VBool False),(L 1,VNum 3)]`
`Nothing`

■ `update::LDir->Val->Mem->Mem`

Actualiza la celda de una memoria con un valor dado. En caso de que la dirección de memoria no se encuentre en la memoria, dicha dirección se agrega.

- `>update (L 3) (VNum 4) [(L 0,VBool True)]`
`[(L 0,VBool True),(L 3,VNum 4)]`
- `>update (L 0) (VBool False) [(L 0,VBool True),(L 1,VNum 3)]`
`[(L 0,VBool False),(L 1,VNum 3)]`

■ `eval1::(Mem,EAB)->(Mem,EAB)`

Extienda la función *eval1*, definida en la práctica 1, al paradigma imperativo.

```
>eval1 ([(L 1, VNum 4)],Suma (VNum 2) (VNum 3))  
([(L 1,VNum 4)],VNum 5)
```

■ `evals::(Mem,EAB)->(Mem,EAB)`

Extienda la función *evals*, definida en la práctica 1, al paradigma imperativo.

```
>evals ([(L 1, VNum 4)], Prod (Suma (VNum 2) (VNum 3)) (Deref $ L 1))  
([(L 1,VNum 4)],VNum 20)
```

■ `interp::EAB->EAB`

Función que interpreta un expresión comenzando con una memoria vacía.

```
>interp $ Let "x" (Ref $ VNum 3) (Suma (VNum 4) (Deref $ Var "x"))  
VNum 7
```

Adicionalmente, deben dar una implementación imperativa, usando los términos del lenguaje *EAB* definidos arriba, de una función que diga si un entero es par, del factorial, de la división, del residuo de una división, del máximo común divisor y una función que simule el *swap*, es decir, que intercambie el contenido de dos referencias. Sus funciones deben ser ejecutadas por la función *interp*.

2. Semántica estática

Para la verificación de tipos con referencias, los contextos son de la forma:

$$\Gamma \mid \Sigma \vdash e : T$$

donde Γ es un conjunto cuyos elementos son de la forma $x : T$, y Σ es un conjunto cuyos elementos son de la forma $\ell : T$. En Haskell representaremos los contextos y los tipos así:

```
type Ctx = [(String,Tipo)],[(LDir,Tipo)]
data Tipo = TInt | TBool deriving (Show,Eq)
```

Implemente la función `vt::Ctx->EAB->Tipo`, que realiza la verificación de tipos de una expresión *EAB* siguiendo las reglas de tipado que vienen en la Nota 3.

```
> vt([],[]) $ Let "x" (Ref $ VNum 3) (Suma (VNum 4) (Deref $ Var "x"))
TInt
```

Además, deberán incluir cinco expresiones de prueba que corran con la función *vt*. Deben usar todos los constructores de *EAB*.