Facultad de Ciencias - UNAM
Lenguajes de Programación 2022-2
Práctica 1: Lenguaje EAB

Favio E. Miranda Perea
Javier Enríquez Mendoza
Ramón Arenas Ayala
Oscar Fernando Millan Pimentel

24 de Marzo de 2022
Fecha de entrega: 5 de Abril de 2022

1 Introducción

En esta práctica implementaremos el lenguaje de Expresiones Aritméticas y Booleanas (EAB) de la nota 4 del curso.

1. (1.5 pts) Crear un nuevo tipo de dato el cual representa la sintaxis abstracta para el lenguaje EAB. Este debe contener las expresiones de:
   - Variables (Var).
   - Números Naturales (Nat).
   - Operadores Aritméticos: suma(Sum), producto(Prod), negación(Neg), predecesor(Pred) y sucesor(Suc).
   - Operadores Booleanos: And, Or, Not e Iszero.
   - Estructuras: if ternario (If) y Let.
   - Constructor para representar la abstraccion (x.e): Abs

2 Semántica Dinámica

Implementaremos una semántica dinámica para nuestro lenguaje de EAB, siento esta una semántica operacional estructural, o sea de paso pequeño. Para esto, se debe hacer de forma apegada a lo visto en las notas de clase, además reconocer expresiones bloqueadas y valores.

1. (3 pts) Implementar la función `eval1 :: EAB` $\rightarrow$ `EAB` tal que eval1 e = e' syss e $\rightarrow$ e'
2. (1 pt) Implementar la función `evals :: EAB` $\rightarrow$ `EAB` tal que evals e e' syss e $\rightarrow^*$ e' y e' está bloqueado.

- **Ejemplo 1:** `evals ((2*6)+true)` debe devolver `12+true`
- **Ejemplo 2:** `evals ((2*6)+3)` debe devolver `15`
- **Ejemplo 3:** `evals (isZero (And True False))` debe devolver `isZero False`

3. (1 pt) Implementar la función `eval :: EAB → EAB` tal que eval e = e' syss e $\rightarrow^*$ e' y e' es un valor La diferencia con evals es que deben manejarse los errores de ejecución.
   - **Ejemplo 1:** `eval(2+true)` debe devolver un error informativo acerca de que la suma solo funciona con números.
   - **Ejemplo 2:** `eval ((2*6)+3)` debe devolver `15`

3 Semántica Estática

Aquí implementaremos la semántica estática para verificación de tipos vista en la nota, y la agregaremos a la evaluación de nuestro lenguaje.

1. (.5 pt) Definir un tipo `Type` para los Tipos de EAB y definir otro tipo `Ctx` para los Contextos
2. (3 pts) Definir la función de verificación de tipado `vt :: Ctx → EAB → Type → Bool` tal que vt Γ e T = True si y sólo si $\Gamma \vdash$ e : T.
   - **Ejemplo 1:** `vt [] 4+5 Num = True`
   - **Ejemplo 2:** `vt [] 4+False Num = False`
   - **Ejemplo 3:** `vt [(x,Num)] 4+x Num = True`
3. (1 pt) Implementar la función de evaluación `evalt :: EAB → EAB` que es esencialmente una reimplementación de la función eval de forma que evalt verifique el tipado de la expresión y sólo en caso positivo inicie el proceso de evaluación. En otro caso debe informarse de un error por existencia de variables libres o por tipado.

4 Extra.

1. (2 pts) Agrega a nuestro lenguajes las expresiones de matchNat y orden "mayor que" (GT) y "menor que" (LT). La especificacion para los operadores de orden es la usual, y para matchNat es la siguiente:
   - **Sintaxis Concreta:** `matchNat e with 0 → e1 | suc x → e2 end`
   - **Semántica:** para evaluar una expresión `matchNat` debemos evaluar e a un valor v. Si v es 0 se devuelve el valor de e1 y si v es suc v' entonces se devuelve el valor de e2 pasando el valor v' a x.
   - **Ejemplo:** la expresion `matchNat pred(5 + 7) with 0 → 7 | suc x → 3*x end` se evalua a 30.

NOTA: En los ejemplos de esta práctica se usa un abuso de notación utilizando Sintaxis Concreta para las expresiones. Esto solo es para facilitar el entendimiento de los ejemplos, pero al utilizar

las funciones se debe utilizar el tipo de dat EAB que representa la sintaxis abstracta de nuestro lenguajes.