Lenguajes de programación 2016-2
Práctica 2

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

Facultad de Ciencias UNAM

1. Inferencia de tipos

El mecanismo de inferencia de tipos es una característica que muchos lenguajes de programación funcional utilizan para calcular el tipo de una expresión que no tiene anotaciones de tipos explícitas.

Por ejemplo, en Haskell el comando `:t` realiza la inferencia de tipos:

```
>:t (\f ->\x ->\y->f (x+y))
(\f ->\x ->\y->f (x+y)) :: Num a =>(a ->r) ->a ->a ->r
```

El objetivo de esta práctica es implementar la inferencia de tipos en el lenguaje *LamAB*, que consiste de expresiones aritméticas, booleanas y funciones.

El lenguaje *LamAB* sin anotaciones de tipos:

```
data LamAB = VNum Int
| VBool Bool
| Var String
| Suma LamAB LamAB
| Prod LamAB LamAB
| Ifte LamAB LamAB LamAB
| Iszero LamAB
| Lam String LamAB
| App LamAB LamAB deriving Show
```

con anotaciones de tipos:

```
data LamABT = VNumT Int
| VBoolT Bool
| VarT String
| SumaT LamABT LamABT
| ProdT LamABT LamABT
| IfteT LamABT LamABT LamABT
| IszeroT LamABT
| Lam String Tipo LamABT
| AppT LamABT LamABT deriving Show
```

1.1. El algoritmo $\mathcal{W}$

El objetivo es recibir una expresión `e::LamAB` y devolver una expresión `e'::LamABT` junto con un contexto Γ y un tipo `T` de tal manera que $\Gamma \vdash$`e':T`.

Definición: La función $\mathcal{W}$ se define recursivamente como sigue:

Variables: $\mathcal{W}(x)$ devuelve $\{x : X\} \vdash x : X$, con X una variable de tipo nueva.

Aplicación: $\mathcal{W}(e_1 e_2)$ devuelve $(\Gamma_1 \cup \Gamma_2)_\mu \vdash (e_1^a e_2^a)_\mu : X_\mu$, donde:

- $\mathcal{W}(e_1)$ devuelve $\Gamma_1 \vdash e_1^a : \mathsf{T}_1$
- $\mathcal{W}(e_2)$ devuelve $\Gamma_2 \vdash e_2^a : \mathsf{T}_2$
- X es una variable de tipo nueva.
- $\mu = umg(\{\mathsf{S}_1 = \mathsf{S}_2 | x : \mathsf{S}_1 \in \Gamma_1, x : \mathsf{S}_2 \in \Gamma_2\} \cup \{\mathsf{T}_1 = \mathsf{T}_2 \to X\})$

Abstracción: Para $\lambda x.e$ primero calculamos $\mathcal{W}(e) = \Gamma \vdash e^a : \mathsf{S}$, entonces:

- Si Γ tiene una declaración para x, digamos $x : \mathsf{R}$, entonces $\mathcal{W}(\lambda x.e)$ devuelve:

$$\Gamma \backslash \{x : \mathsf{R}\} \vdash \lambda x : \mathsf{R}.e^a : \mathsf{R} \to \mathsf{S}$$

- Si Γ no tiene una declaración para x, $\mathcal{W}(\lambda x.e)$ devuelve:

$$\Gamma \vdash \lambda x : X.e^a : X \to \mathsf{S}$$

  con X una variable de tipo nueva.

Extendemos la función $\mathcal{W}$ a $LamAB$ como sigue:

Suma: $\mathcal{W}(e_1 + e_2)$ devuelve $(\Gamma_1 \cup \Gamma_2)_\mu \vdash (e_1^a + e_2^a)_\mu : \mathsf{Nat}$, donde:

- $\mathcal{W}(e_1)$ devuelve $\Gamma_1 \vdash e_1^a : \mathsf{T}_1$
- $\mathcal{W}(e_2)$ devuelve $\Gamma_2 \vdash e_2^a : \mathsf{T}_2$
- $\mu = umg(\{\mathsf{S}_1 = \mathsf{S}_2 | x : \mathsf{S}_1 \in \Gamma_1, x : \mathsf{S}_2 \in \Gamma_2\} \cup \{\mathsf{T}_1 = \mathsf{Nat}, \mathsf{T}_2 = \mathsf{Nat}\})$

análogamente para las demás expresiones, ajenas al cálculo lambda, de $LamAB$.

Para esto, usaremos el tipo:

```
data Tipo = TNat | TBool | X Nombre | Tipo:->Tipo deriving Eq
```

para representar nuestros tipos.

Tomamos el contexto de un juicio de tipado como: **type** `Ctx = [(String, Tipo)]`, y los juicios de tipado como: **data** `Juicio = Deriv (Ctx,LamAB,Tipo)`.

Para realizar la implementación del algoritmo $\mathcal{W}$, en la sesión de laboratorio se discutió el procedimiento para obtener variables nuevas, así como la tarea de unificación para determinar el unificador más general μ, y las sustituciones sobre las que μ tiene efecto.

1.2. Ejercicio:

- `algoritmoW :: LamAB->Juicio`

  Realiza la inferencia de tipos de una expresión `LamAB`.

  Aquí algunos ejemplos de ejecución:

  - `>algoritmoW $ Lambda "x" $ Var "x"`
    `[]:-LambdaT "x" X0 (Var x):X0->X0`
  - `>algoritmoW $ Lambda "x" $ Var "y"`
    `[("y",X0)]:-LambdaT "x" X1 (Var y):X1->X0`
  - `>algoritmoW $ Lambda "x" $ App (Var "y") (Var "z")`
    `[("y",X0->X1),("z",X0)]:-LambdaT "x" X2 (App (VarT "y") (VarT "z")):X2->X1`
  - `>algoritmoW $ LambdaE "x" $ App (Var "x") (Var "x")`
    `*** Exception: No se pudo unificar.`
  - `>algoritmoW $ App (Var "y") (Lambda "x" $ Var "x")`
    `[("y",(X0->X0)->X1)]:-AppT (VarT "y") (LambdaT "x" X2 $ VarT "x"):X1`
  - `>algoritmoW $ App (Var "x") (Suma (VNum 1) (VNum 3))`
    `["x",Nat->X0]:-AppT (VarT "x") (SumaT (VNumT 1) (VNumT 3)):X0`
  - `>algoritmoW $ App (VNum 0) (VNum 3)`
    `*** Exception: No se pudo unificar.`