Facultad de Ciencias - UNAM
Lenguajes de Programación 2022-2
Práctica 2: Algoritmo de inferencia de tipos

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

04 de Mayo de 2022
Fecha de entrega: 17 de Mayo de 2022

1 Mini Haskell (MinHs)

MinHs es un pequeño lenguaje de programación que implementa los conceptos y mecanismos esenciales del paradigma de programación funcional vistos hasta ahora en clase.
Las expresiones de este lenguaje son:

```
data Expr = V Identifier | I Int | B Bool
.          | Fn Identifier Expr
.          | Succ Expr | Pred Expr
.          | Add Expr Expr | Mul Expr Expr
.          | Not Expr | Iszero Expr
.          | And Expr Expr | Or Expr Expr
.          | Lt Expr Expr | Gt Expr Expr | Eq Expr Expr
.          | If Expr Expr Expr
.          | Let Identifier Expr Expr
.          | App Expr Expr
```

En la práctica anterior implementamos la semántica dinámica y estática para el lenguaje de expresiones aritméticas y booleanas (EAB). Para tener nuestro lenguaje `MinHs` debemos extender EAB agregando los constructores `Fn`, `App`, `Lt`, `Gt` y `Eq`. *(solo hay que agregarlos al tipo de dato, para esta práctica no es necesario modificar funciones de la anterior)*

2 Algoritmo de inferencia de tipos

2.1 La relación de tipado con restricciones

El algoritmo consta de dos pasos, el primero consiste en construir una derivación de tipos con restricciones. Para este propósito definimos una relación cuaternaria $R \mid \Gamma \vdash \mathrm{e}:\ \mathrm{T}$, cuyo significado intuitivo es

$\Gamma \models e :\ T$ *se cumple cuando el conjunto de restricciones* R *es satisfacible.*

Las siguientes reglas definen a esta relación y permiten construir sus derivaciones:

- Tipado de variables.

$$\frac{x : T \in \Gamma}{\varnothing|\Gamma \models x : T}$$

- Tipado de constantes.

$$\frac{}{\varnothing|\varnothing \models n : Integer}$$

$$\frac{}{\varnothing|\varnothing \models b : Boolean}$$

- Tipado de funciones.

$$\frac{R|\Gamma, x : S \models e : T}{R|\Gamma \models fn(x.e) : S \to T}$$

- Tipado de operadores unarios.

$$\frac{R|\Gamma \models e : S}{R, S = Integer|\Gamma \models succ(e) : Integer}$$

  Similar para pred() y not().

- Tipado de operadores n-narios.

$$\frac{\begin{array}{c} R_1|\Gamma_1 \models e1 : T1 \\ R_2|\Gamma_2 \models e2 : T2 \\ tVars(R_1 \cup \Gamma_1 \cup T1) \cap tVars(R_2 \cup \Gamma_2 \cup T2) = \varnothing \\ S = \{S1 = S2 | x : S1 \in \Gamma_1, x : S2 \in \Gamma_2\} \end{array}}{R_1, R_2, S, T1 = Integer, T2 = Integer|\Gamma_1, \Gamma_2 \models add(e1, e2) : Integer}$$

  Similar para mul(), and(), or(), lt(), gt(), eq() e if().

- Tipado de aplicaciones.

$$\frac{\begin{array}{c} R_1|\Gamma_1 \models e1 : T1 \\ R_2|\Gamma_2 \models e2 : T2 \\ tVars(R_1 \cup \Gamma_1 \cup T1) \cap tVars(R_2 \cup \Gamma_2 \cup T2) = \varnothing \\ S = \{S1 = S2 | x : S1 \in \Gamma_1, x : S2 \in \Gamma_2\} \\ Zfresh \end{array}}{T1 = T2 \to Z, S, R_1, R_2|\Gamma_1, \Gamma_2 \models e1\, e2 : Z}$$

  donde Z fresh significa que Z es una variable de tipo nueva, es decir, una variable que no figura en ninguna de las premisas anteriores de la regla.

- Tipado de expresión let.v a r s (T 1 :> (T 2 :> T 1)) [1 , 2] Main> v a r s (Integer :> (Boolean :> Integer)) [ ]

$$\frac{\begin{array}{c} R_1|\Gamma_1 \models e1 : T1 \\ R_2|\Gamma_2, x : S \models e2 : T2 \\ tVars(R_1 \cup \Gamma_1 \cup T1) \cap tVars(R_2 \cup \Gamma_2 \cup T2) = \varnothing \\ S = \{S1 = S2 | x : S1 \in \Gamma_1, x : S2 \in \Gamma_2\} \end{array}}{R_1, R_2, S, T1 = S|\Gamma_1, \Gamma_2 \models let(e1, x.e2) : T2}$$

Para realizar la implementación de esto, debemos definir los siguientes tipos de dato:

```
type Identifier = Int
data Type = T Identifier
.          | Integer | Boolean
.          | Arrow Type Type
```

Este contexto es igual al que utilizamos en la practica anterior para verificacion de tipos.

`type Ctxt = [ ( Identifier , Type ) ]` *Identifier definido en el modulo de la sintaxis.*

Por ultimo definimos un conjunto de restricciones como sigue.

`type Constraint = [ ( Type , Type ) ]`

Tambien debemos realizar lo siguiente:

1. (1 pt) Implementar la función `tvars :: Type → [Identifier]` la cual devuelve el conjunto de variables de tipo.
   Ejemplos:
   `main > tvars ( T1 → ( T2 → T1 ) ) ⇒ [1,2]`
   `main > tvars ( Integer → ( Boolean → Integer ) ) ⇒ []`

2. (1 pt) Implementar la función `fresh :: [Type] → Type` la cual dado un conjunto de variables de tipo, obtiene una variable de tipo fresca, es decir, que no aparece en este conjunto. **Esta funcion es importante que se realice utilizando lo discutido durante el laboratorio sobre el problema de MinFree.**
   Ejemplos:
   `main > fresh [T0 , T1 , T2 , T3] ⇒ T4`
   `main > fresh [T0 , T1 , T3 , T4] ⇒ T2`

3. (1 pt) Implementar la función `rest :: ( [ Type ] , Expr ) → ( [ Type ] , Ctxt , Type , Constraint )` la cual dada una expresion, infiere su tipo implementando las reglas descritas anteriormente. Devolviendo el contexto y el conjunto de restricciones donde es valido. Utiliza el conjunto de variables de tipo para crear variables de tipo frecas durante la ejecucion.
   Ejemplos:
   `main > rest ( [] , Fn ”x” (V ”x” ) ) ⇒`
   `( [ T0 ] , [ ] , T0 > T0 , [ ] )`
   `main > rest ( [] , Add (V ”x” ) (V ”x” ) ) ⇒`
   `( [ T0 , T1 ] , [ ( ”x” , T0 ) , ( ”x” , T1 ) ] , Integer , [ ( T0 , T1 ) , (T0 , Integer ) , (T1 , Integer ) ] )`

2.2 Algoritmo de unificacion *U*

La segunda parte del algoritmo de inferencia debemos intentar resolver el conjunto de restricciones para obtener el tipo de la expresion. Esto es obtener el unificador mas general (μ).

Para esto utilzaremos la definicion del algoritmo que esta en la nota 7, asi como la definicion de substitucion y de composicion de esta.

```
 Definimos como una lista de duplas la substitucion en tipos.
type Substitution = [ ( Identifier , Type ) ]
```

1. (1 pt) Implementar la función `subst :: Type → Substitution → Type` la cual aplica la sustitucion a un tipo dado.
   Ejemplos:
   `main > subst (T1 → (T2 → T1)) [(3 , T4) , (5 , T6)] ⇒ T1 → (T2 → T1)`
   `main > subst (T1 → (T2 → T1)) [(1 , T2) , (2 , T3)] ⇒ T2 → (T3 → T2 )`

2. (1 pt) Implementar la función `comp :: Substitution → Substitution → Substitution` la cual realiza la composicion de dos sustituciones.
   Ejemplos:
   `main > comp [(1 , T2 → T3) , (4 , T5)] [( 2 , T6)] ⇒ [(1 , T6 → T3) , (4 , T5) , (2 , T6)]`

3. (1 pt) Implementar la función `unif :: Constraint` $\rightarrow$ `Substitution` la cual obtiene el unificador mas general (μ). Pueden consultar la implementacion realizada durante el laboratorio.

2.3 Inferencia de tipos.

1. (1 pt) Implementar la función `infer :: Expr > ( Ctxt , Type )` la cual dada una expresion, infiere su tipo devolviendo el contexto donde es valido.
   Ejemplos:
   `main > infer ( Let ”x” (B True) (And (V ”x” ) ( Let ”x” ( I 1 0 ) (Eq ( I 0 ) ( Succ (V ”x” )))))) ⇒ ( [ ] , Boolean )`

3 Especificaciones de entrega y notas.

- **Adjuntos enviados:** Realizar la entrega en un comprimido llamado 'Nombre_del_equipo.zip', donde el nombre de su equipo es el que ustedes quieran. Esta carpeta debe contener todos los archivos de su practica y un archivo con el nombre de los integrantes del equipo.
- **Entrega en el Classroom:** Solo un integrante del equipo debe realizar la entrega, los demas no es necesario que pongan como completada la practica.
- **Nota:** en los ejemplos se utiliza un abuso de notacion tanto por facilidad como para que sea mas entendible, al querer usar sus funciones deben usar la sintaxis correcta.

Buena suerte a todos, cualquier duda pueden preguntar preferentemente a traves del *Telegram* del grupo ! ☺