Facultad de Ciencias - UNAM
Lógica Computacional 2023-1
Práctica 3 - Implementación del algoritmo de unificación Martelli-Montanari

Favio E Miranda Perea
Javier Enríquez Mendoza
José Manuel Madrigal Ramírez

18/octubre/2022
Fecha de entrega: 31/octubre/2022 a las 23:59 hrs

Introducción

El objetivo de esta práctica es implementar el algoritmo de unificación *Martelli-Montanari* cuya entrada es un conjunto de ecuaciones y su salida es el unificador más general μ en caso de que los términos en las ecuaciones fuesen unificables o, en su defecto, un mensaje de error.

En el archivo `unifica.hs` están las firmas de tipo de las funciones que implementaremos así como varias definiciones que nos serán de utilidad:

- `Ident`: son los identificadores de nuestros términos y están representados como alias del tipo `String`.

```
type Ident = String
```

- `Termino`: un tipo de dato para los términos con dos constructores, uno para las variables y el otro para las funciones con su respectiva lista de argumentos que, a su vez, son términos.

```
data Termino = V Ident
             | T Ident [Termino]
    deriving Eq
```

- `Variable`: un alias para `Termino`, que nos brindará claridad al escribir las firmas de tipo de las funciones que definiremos.

```
type Variable = Termino
```

- `Sustitucion`: una lista de pares formados por variables y términos. Por ejemplo, la sustitución $\rho = [x := a, z := f(x, y)]$ quedaría representada como `[(x, a), (z, f [x, y])]`:

```
type Sustitucion = [(Variable, Termino)]
```

- Una instancia de *show* para `Termino`.

```
instance Show Termino where
 show (V ident)    = ident
 show (T ident []) = ident
 show (T ident ts) = ident ++ concat [ show ts]
```

- Una miscelánea de ejemplos de los cuales se servirán, a su vez, los ejemplos de esta práctica:

  - 3 símbolos de constantes a, b y c

    Notemos que las constantes se implementan como términos sin argumentos. De este modo la constante a queda implementada como:

```
a = T "a" []          --Constructor T
                      --Identificador "a"
                      --Lista de términos vacía []
```

  - Variables:

```
x = V "x"
y = V "y"
z = V "z"
u = V "u"
```

  - Símbolos de función:

```
f = T "f"
g = T "g"
h = T "h"
```

  - Términos:

```
t1 = f [x,y,x]
t2 = f [y, g [x],x]
t3 = f [g [x], h [x,u]]
t4 = f [z, h [f [y,y],z]]
t5 = h [g [z]]
t6 = h [f [a], g [x]]
t7 = h [z,z]
t8 = h [y,z]
t9 = h [x, g [a]]
t10 = h [g [z],z]
```

  - Sustituciones:

```
s1 = [(x, a), (z, f [x, y])]
s2 = [(x, z), (y, u)]
s3 = [(z, x), (x, b), (u, c)]
```

```
4       s4 = [(u, f [x]), (y, a)]
5       s5 = [(x, h [z]), (y, g [b])]
6       s6 = [(z, g [x])]
7       s7 = [(x, f [y,y])]
8       s8 = [(u, g [f [y,y]])]
```

En los ejemplos de las secciones subsecuentes se harán referencia

Funciones auxiliares sobre términos

1. Comencemos por definir una función `esVariable` que reciba un término y regrese un booleano para indicar si dicho término es, o no, una variable. Veamos algunos ejemplos:

```
*Unificacion>esVariable a
False
*Unificacion>esVariable x
True
*Unificacion>esVariable t1
False
```

2. Ahora definamos una función que, dado un término, regrese la lista de variables que figuren en él, sin repeticiones. Aquí unos ejemplos de su ejecución:

```
*Unificacion>variables t3
[x,u]
*Unificacion>variables a
[]
*Unificacion>variables y
[y]
```

3. Escribamos una función que dada una lista de términos, regrese la lista de variables que figuren en ellos. Veamos algunos casos particulares de su ejecución:

```
*Unificacion>variablesEnLista [t1, t2, t3]
[x,y,u]
*Unificacion>variablesEnLista []
[]
```

   Pista: `nub` es una función predefinida que elimina elementos repetidos en una lista.

Funciones auxiliares sobre sustituciones

1. Definamos una función que regrese el dominio de una sustitución, es decir, el conjunto de variables a reemplazar. Recuerda que en los ejemplos `s1` y `s3` corresponden a las sustituciones definidas en la sección de *introducción* y que también se encuentran en el archivo `unifica.hs`:

```
*Unificacion>dominio s1
[x,z]
*Unificacion>dominio s3
[z,x,u]
```

2. Ahora escribamos una función que aplique una sustitución a una variable. La idea es que regresemos el término correspondiente si es que la variable de entrada coincide con alguna variable presente en el dominio de la sustitución. Y si no coincide, devolvemos la variable original.
   En el siguiente ejemplo z figura en el dominio de `s1` mientras que u no se encuentra en el dominio de dicha sustitución:

```
*Unificacion>aplicaVar s1 u
u
*Unificacion>aplicaVar s1 z
f[x,y]
```

3. Definamos una función que tome una sustitución y la aplique a un término dado, para ello podemos utilizar la función anterior que aplicaba la sustitución a una variable. Veamos algunos ejemplos de su ejecución:

```
*Unificacion>aplicaT s3 t1
f[b,y,b]
*Unificacion>aplicaT s5 t9
h[h[z],g[a]]
```

1. Composición de sustituciones

1. Es posible que en el proceso de componer dos sustituciones el lado derecho de un par sea igual al lado izquierdo. Es decir, que se indique la sustitución de una variable por ella misma:

   $$[u := u]$$

   Dada la definición de composición de sustituciones en las notas de clase 5, en la sección 3.2, debemos evitar dichos casos. Definamos una función que tome una sustitución y elimine los pares cuyos elementos sean iguales. Por ejemplo, esperaríamos que la sustitución

   $$\rho = [x := a, z := f(x, y), y := y, u := u]$$

   pudiera reducirse a:

   $$\rho = [x := a, z := f(x, y)]$$

   como sigue:

```
*Unificacion>rho
[(x,a),(z,f[x,y]),(y,y),(u,u)]
*Unificacion>reduce rho
[(x,a),(z,f[x,y])]
```

2. Hecho lo anterior, vamos a implementar la composición de sustituciones, no sin antes ver un ejemplo. Si tenemos las siguientes sustituciones:

$$\sigma = [x := a, z := f(x, y)] \qquad \rho = [x := z, y := u]$$

Primero aplicamos ρ a los términos del lado derecho de σ:

$$[x := a\rho, z := f(x, y)\rho]$$

$$[x := a, z := f(z, u)]$$

Si hubiese resultado un término idéntico a la variable del lado izquierdo, eliminaríamos ese par de la sustitución.

A continuación añadimos los pares de ρ:

$$[x := a, z := f(z, u), x := z, y := u]$$

Pero debemos evitar añadir aquellos pares que empiecen con la misma variable que algún par de σ, así que quitamos el par $x := z$ obteniendo de este modo la composición:

$$\sigma\rho = [x := a, z := f(z, u), y := u]$$

De aquí podemos rescatar algunas pautas generales:

- Aplicamos ρ a los términos del lado derecho de σ.
- Eliminamos los pares cuyos elementos hayan quedado idénticos tras haber realizado la aplicación anterior (cosa que no sucedió en este ejemplo).
- Añadimos las sustituciones de ρ excepto aquellas que tuvieran una variable que ya figurase en el lado izquierdo de algún par de σ.

Ahora veamos un par de ejemplos de la ejecución de la composición de sustituciones en terminal:

```
*Unificacion>s1
[(x,a),(z,f[x,y])]
*Unificacion>s2
[(x,z),(y,u)]
*Unificacion>composicion s1 s2
[(x,a),(z,f[z,u]),(y,u)]
```

```
*Unificacion>s3
[(z,x),(x,b),(u,c)]
*Unificacion>s4
[(u,f[x]),(y,a)]
*Unificacion>composicion s3 s4
[(z,x),(x,b),(u,c),(y,a)]
```

3. Finalmente vamos a implementar una función que componga todas las sustituciones de una lista dada:

```
*Unificacion>s2
[(x,z),(y,u)]
*Unificacion>s6
[(z,g[x])]
*Unificacion>s7
[(x,f[y,y])]
*Unificacion>subs = [s2, s6, s7]
*Unificacion>complista subs
[(x,g[f[y,y]]),(y,u),(z,g[f[y,y]])]
```

El algoritmo Martelli-Montanari

Antes de comenzar, observemos que una ecuación es un par ordenado de términos:

```
type EqTerm = (Termino,Termino)
```

1. Definamos una función que aplique una sustitución a una ecuación. Así, tendríamos:

   ```
   *Unificacion>eq1 = (t1,t3)
   *Unificacion>eq1
   (f[x,y,x],f[g[x],h[x,u]])
   *Unificacion>s1
   [(x,a),(z,f[x,y])]
   *Unificacion>apSustEq s1 eq1
   (f[a,y,a],f[g[a],h[a,u]])
   ```

2. Al fin ha llegado el momento de escribir la función que implementará el algoritmo de Martelli-Montanari. Esta función toma una lista de ecuaciones, una lista de sustituciones y regresa el unificador más general μ. Dicha lista de sustituciones esta compuesta por aquellas sustituciones que van surgiendo cuando aplicamos la regla [Sust], como viene especificado en la sección 5.3 de las notas de clase 5.

   Consideremos como caso base el momento en que la lista de ecuaciones esté vacía. En ese momento habrá llegado el momento de componer las sustituciones en la lista para generar el unificador más general.

   ```
   unifmm [] subs = ...
   ```

   El resto de casos estarán divididos entre las reglas [Sust], [Desc] y [Swap]. En caso de que la descomposición o la sustitución fallen, podemos regresar un mensaje de *error*.

   ```
   unifmm ((V x,t2):eqs) subs = ...
   unifmm ((t1,V x):eqs) subs = ...
   unifmm ((T f ts, T g rs):eqs) subs = ...
   ```

   Veamos algunos ejemplos de esta función:

```
**Unificacion>t1
*f[x,y,x]
**Unificacion>t2
*f[y,g[x],x]
**Unificacion>eq2 = (t1,t2)
**Unificacion>unifmm [eq2] []
**** Exception: La ecuacion y = g[y] no es unificable
*CallStack (from HasCallStack):
* error, called at unifica.hs:119:32 in main:Unificacion
```

```
**Unificacion>t6
*h[f[a],g[x]]
**Unificacion>t7
*h[z,z]
**Unificacion>eq3 = (t6,t7)
**Unificacion>unifmm [eq3] []
**** Exception: Los símbolos de función "gz "f"son distintos
*CallStack (from HasCallStack):
* error, called at unifica.hs:131:16 in main:Unificacion
```

```
**Unificacion>t6
*h[f[a],g[x]]
**Unificacion>t8
*h[y,z]
**Unificacion>eq4 = (t6, t8)
**Unificacion>unifmm [eq4] []
*[(y,f[a]),(z,g[x])]
```

3. Por último, definamos una función `unifL` que tome un conjunto de términos y regrese el unificador más general en caso de que dicho conjunto sea unificable.

```
*Unificacion>t3
f[g[x],h[x,u]]
*Unificacion>t4
f[z,h[f[y,y],z]]
*Unificacion>unifL [t3,t4]
[(z,g[f[y,y]]),(x,f[y,y]),(u,g[f[y,y]])]
```

```
*Unificacion>t9
h[x,g[a]]
*Unificacion>t10
h[g[z],z]
*Unificacion>unifL [t9,t10]
[(x,g[g[a]]),(z,g[a])]
```

Pista: Podemos fijarnos, de entrada, en el primer par de términos de la lista l, digamos *t1* y *t2*. Llamamos a Martelli-Montanari sobre ellos. Si obtenemos un unificador más general μ, lo aplicamos a *t1* y *t2*. Añadimos el resultado al resto de la lista ls de términos para obtener l_2. Aplicamos `unifL` a esa nueva lista l_2 y así sucesivamente.

Entrega

La entrega se realizará por **parejas** y consistirá en un archivo comprimido que debe contener:

- El archivo `unifica.hs` con el código necesario para que sus funciones se ejecuten adecuadamente.
- Un archivo readMe.txt que incluya sus nombres y una bitácora sobre el desarrollo de la práctica donde indiquen los problemas que se presentaron y cómo los resolvieron. Asimismo incluyan comentarios de cada uno al respecto.

Envíen su archivo comprimido a mi correo: *jose.manuel.madrigal.ramirez@gmail.com* con el siguiente formato:

Practica3-Apellido1Nombre1Apellido2Nombre2.zip

Por ejemplo, si su equipo esta conformado por Alan Turing y Ada Lovelace su archivo debería llamarse:

Practica3-TuringAlanLovelaceAda.zip

Cualquier duda que tengan no duden en enviarme un correo o comunicarse por Telegram. 😄