Lógica Computacional, 2019-2
Nota de laboratorio 8*

Sustitución en Lógica de Primer Orden

Manuel Soto Romero

27 de marzo de 2019
Facultad de Ciencias, UNAM

La noción de sustitución en la Lógica de Primer Orden es más complicada que en la Lógica Proposicional debido a la existencia de términos y variables ligadas. A diferencia de lo que sucede en la Lógica Proposicional, las sustituciones son operaciones sobre términos y sobre las fórmulas y se debe respetar el ligado de las variables, por lo tanto *no* son operaciones textuales. En esta nota se revisan los detalles y retos de la implementación de esta operación.

1. Sustitución

Definición 1. Una *sustitución* en un lenguaje de primer orden L es una tupla de variables y términos denotada como $[x_1, ..., x_n := t_1, ..., t_n]$ dónde:

- $x_1, ..., x_n$ son variables distintas.
- $t_1, ..., t_n$ son términos de L.
- $x_i \neq t_i$ para toda $1 \leq i \leq n$.

Se denota a una sustitución con $\left[\vec{x} := \vec{t}\right]$.

Para representar sustituciones en Haskell, se hace uso de listas de pares mediante el sinónimo `Subst` que se muestra en el Listado de código 1.

```haskell
-- Definición de sustituciones
type Sust = [(String,TERM)]
```

Listado de código 1: Sinónimo para representar sustituciones

Para verificar que una sustitución dada sea legal, esto es que cumpla con la Definición 1, se requiere de una función `verifSus :: Sust -> Bool` que se define en el Listado de código 2.

*Basado en las *Notas de clase para el curso de Lógica Computacional* del Dr. Favio E. Miranda, et al., revisión 2017-2.

```haskell
-- Verifica si una sustitución es legal.
verifSus :: Sust -> Bool
verifSus [] = True
verifSus ls = ver1 ls && ver2 ls

-- Verifica que una lista no tiene elementos repetidos.
ver1 :: [(String,TERM)] -> Bool
ver1 []     = True
ver1 (x:xs) = not (elems x xs) && ver1 xs

-- Busca en una lista de tuplas.
elems :: (String,TERM) -> [(String,TERM)] -> Bool
elems _ [] = False
elems (x,y) ((w,_):xs) = x == w || elems (x,y) xs

-- Verifica que una variable no se sustituya consigo misma.
ver2 :: [(String,TERM)] -> Bool
ver2 [] = True
ver2 ((x,(Var v)):xs) = x /= v && ver2 xs
ver2 (x:xs) = ver2 xs
```

Listado de código 2: Verificación de sustituciones

Observación 1. La función `ver1` (líneas 7 a 9) verifica que todas las variables de una sustitución sean distintas, mientras que la función `var2` (líneas 17 a 29) verifica que no se sustituya una variable consigo misma. El sistema de tipos de Haskell se encarga de verificar que el lado derecho de una sustitución en efecto es un término.

2. Sustitución de términos

La aplicación de una sustitución $[\vec{x}:=\vec{t}]$ a un término r, denotada $r[\vec{x}:=\vec{t}]$, se define como el término obtenido al reemplazar *simultáneamente* todas las presencias de x_i en r por t_i. Este proceso se define recursivamente como sigue:

$$x_i[\vec{x}:=\vec{t}] = t_i \;\; si \;\; 1 \leq i \leq n$$
$$z[\vec{x}:=\vec{t}] = z \;\; si \;\; z \neq x_i, 1 \leq i \leq n$$
$$c[\vec{x}:=\vec{t}] = c \;\; si \;\; c \in C$$
$$f\,(t_1,...,t_m)\,[\vec{x}:=\vec{t}] = f\left(t_1[\vec{x}:=\vec{t}],...,t_m[\vec{x}:=\vec{t}]\right) \;\; con \;\; f^{(m)} \in F$$

El Listado de código 3 muestra la implementación de esta definición.

```haskell
-- Sustitución de términos.
apsusT :: TERM -> Sust -> TERM
apsusT t []           = t
apSust (Var v) ys     = busca v ls
apSust (Fun c []) _   = Fun c []
spSust (Fun f xs) ys = Fun f (map (\x -> apSust x ys) xs)

-- Busqueda de variables.
busca :: String -> Sust -> TERM
busca _ []          = error "Lista vacía"
busca s ((x,y):xs)
   | s == x = y
   | otherwise = busca s xs
```

Listado de código 3: Sustitución de términos

3. Sustitución parcial de fórmulas

La aplicación de sustitución a fórmulas, necesita de ciertos cuidados debido a la presencia de variables ligadas mediante cuantificadores. Deben evitarse los siguientes problemas[1]:

- Sustitución de variables ligadas.
- Captura de variables libres.

Definición 2. La aplicación de una sustitución $[\vec{x}:=\vec{t}]$ a una fórmula φ, denotada $\varphi[\vec{x}:=\vec{t}]$ se define como la fórmula obtenida al reemplazar simultáneamente todas las presencias libres de x_i en φ por t_i, verificando que este proceso no capture posiciones de variables libres.

La aplicación de una sustitución a una fórmula $\varphi[\vec{x}:=\vec{t}]$ se define *recursivamente* como sigue:

$$\bot[\vec{x}:=\vec{t}] = \bot$$
$$\top[\vec{x}:=\vec{t}] = \top$$
$$P\,(t_1,...,t_m)\,[\vec{x}:=\vec{t}] = P(t_1[\vec{x}:=\vec{t}],...,t_m[\vec{x}:=\vec{t}])$$
$$(t_1 = t_2)[\vec{x}:=\vec{t}] = t_1[\vec{x}:=\vec{t}] = t_2[\vec{x}:=\vec{t}]$$

[1]La explicación de estos problemas se explica a detalle en las *Notas de clase para el curso de Lógica Computaiconal* del Dr. Favio E. Miranda, et. al., revisión 2017-2.

$$(\neg\varphi)\,[\vec{x}:=\vec{t}] = \neg(\varphi[\vec{x}:=\vec{t}])$$
$$(\varphi \wedge \psi)\,[\vec{x}:=\vec{t}] = (\varphi[\vec{x}:=\vec{t}] \wedge \psi[\vec{x}:=\vec{t}])$$
$$(\varphi \vee \psi)\,[\vec{x}:=\vec{t}] = (\varphi[\vec{x}:=\vec{t}] \vee \psi[\vec{x}:=\vec{t}])$$
$$(\varphi \rightarrow \psi)\,[\vec{x}:=\vec{t}] = (\varphi[\vec{x}:=\vec{t}] \rightarrow \psi[\vec{x}:=\vec{t}])$$
$$(\varphi \leftrightarrow \psi)\,[\vec{x}:=\vec{t}] = (\varphi[\vec{x}:=\vec{t}] \leftrightarrow \psi[\vec{x}:=\vec{t}])$$

$$(\forall y\varphi)\,[\vec{x}:=\vec{t}] = \forall y(\varphi[\vec{x}:=\vec{t}]) \;\; si \;\; y \notin \vec{x} \cup VAR(\vec{t})$$
$$(\exists y\varphi)\,[\vec{x}:=\vec{t}] = \exists y(\varphi[\vec{x}:=\vec{t}]) \;\; si \;\; y \notin \vec{x} \cup VAR(\vec{t})$$

El Listado de código 4 muestra la implementación de esta definición.

```
1  -- Sustitución de fórmulas
2  apsusF :: PO -> Sust -> PO
3  apsusF (FA (Cte b))    _  = FA (Cte b)
4  apsusF (FA (Prd n ls) ys = FA (Prd n (map (\x -> apsusT x ys) ls))
5  apsusF (FA (Eq p q)   ys = Fa (Eq (apsusT p ys) (apsusT q ys))
6  apsusF (Neg p)        ys = Neg (apsusF p ys)
7  apsusF (Conj p q)     ys = Conj (apsusF p ys) (apsusF q ys)
8  apsusF (Disy p q)     ys = Disy (apsusF p ys) (apsusF q ys)
9  apsusF (Impl p q)     ys = Impl (apsusF p ys) (apsusF q ys)
10 apsusF (Syss p q)     ys = Syss (apsusF p ys) (apsusF q ys)
11 apsusF (Pt x p)       ys = ...
12 apsusF (Ex x p)       ys = ...
```

Listado de código 4: Sustitución de términos

La implementación de los dos últimos casos se deja como ejercicio al lector.

La definición anterior cuenta con una restricción en el caso de los cuantificadores, por ejemplo, la siguiente sustitución:

$$\forall x\,(Q\,(x) \rightarrow R\,(z, y))\,[z := f\,(x)]$$

no está definida, puesto que $x \in VAR\,(f\,(x))$. Por lo tanto, la aplicación de cualquier sustitución a una fórmula hasta ahora es una función parcial. Sin embargo, se puede apreciar que los nombres de las variables realmente no importan. Por ejemplo, las siguientes fórmulas:

$$\forall x P\,(x) \qquad \forall y P\,(y)$$

significan lo mismo.

Es posible entonces, identificar fórmulas que sólo difieren en sus variables ligadas. Esto se hace mediante una relación de α-*equivalencia*.

4. Relación de α-equivalencia

Definición 3. Se dice que dos fórmulas φ_1, φ_2 son α-equivalentes y se escribe $\varphi_1 \sim_\alpha \varphi_2$ si y sólo sí φ_1 y φ_2 difieren a lo más en los nombres de sus variables ligadas.

Usando α-equivalencias, la operación de sustitución en fórmulas (`apsusF`) se vuelve una función total por lo que siempre está definida.

Ejemplo 1. Realizar la siguiente sustitución:

$$\exists z Q\left(y, z, y\right)\left[y, z := g\left(d\right), f\left(y\right)\right]$$

Solución. Se realizan los siguientes pasos:

1. Se necesita una α-equivalencia, pues $z \in \vec{x}$.
   $\exists z Q\left(y, z, y\right)\left[y, z := g\left(d\right), f\left(y\right)\right] \sim_\alpha \exists w Q\left(y, w, y\right)\left[y, z := g\left(d\right), f\left(y\right)\right]$
2. $\exists w \left(Q\left(y, w, y\right)\left[y, z := g\left(d\right), f\left(y\right)\right]\right)$
3. $\exists w Q\left(y[y, z := g\left(d\right), f\left(y\right)], w[y, z := g\left(d\right), f\left(y\right)], y[y, z := g\left(d\right), f\left(y\right)]\right)$
4. $\exists w Q\left(g\left(d\right), w, g\left(d\right)\right)$

□

La implementación de la función total requiere:

1. Una forma de verificar si dos fórmulas son α-equivalentes.
2. Una forma de renombrar las variables ligadas de una fórmula de forma que las listas de variables libres y ligadas sean ajenas.
3. Una forma de renombrar las variables ligadas de una fórmula de forma que sus nombres sean ajenos a los de una lista dada.