Lógica de Primer Orden: Conceptos Sintácticos Importantes
Lógica Computacional 2018-2, Nota de clase 5

Favio Ezequiel Miranda Perea Araceli Liliana Reyes Cabello
Lourdes Del Carmen González Huesca Pilar Selene Linares Arévalo

6 de marzo de 2018
Facultad de Ciencias UNAM

Al tener cuantificadores en la lógica de primer orden se deben revisar con detenimiento algunos conceptos sintácticos como la importancia de un cuantificador en una fórmula así como los diferentes casos a estudiar al tratar de realizar una sustitución.

1. Ligado y alcance

En lógica de predicados los cuantificadores $\forall x$ y $\exists x$ son ejemplos del fenómeno de *ligado* que también surge en la mayoría de lenguajes de programación. La idea general es que ciertas presencias de variables son presencias *ligadas* o simplemente *ligas*, cada una de las cuales se asocia con una expresión llamada su *alcance*.

Definición 1 *Dada una cuantificación $\forall x\varphi$ o $\exists x\varphi$, la presencia de x en $\forall x$ o $\exists x$ es la variable (artificial) que liga el cuantificador correspondiente, mientras que la fórmula φ se llama el alcance, ámbito o radio de acción de este cuantificador.*

Definición 2 *Una presencia de la variable x en la fórmula φ está ligada o acotada si es la variable que liga a un cuantificador de φ o si figura en el alcance de un cuantificador $\forall x$ o $\exists x$ de φ.*
Si una presencia de la variable x en la fórmula φ no está ligada, decimos que está libre en φ.

Ejemplo 1.1 Sea $\varphi = \forall x \exists z (Q(y,z) \vee R(z,x,y)) \wedge P(z,x)$.

- El alcance del cuantificador $\forall$ es la fórmula $\exists z(Q(y,z) \vee R(z,x,y))$.
- El alcance del cuantificador $\exists$ es la fórmula $Q(y,z) \vee R(z,x,y)$.
- En φ hay tres presencias de x, las dos primeras ligadas y la última libre.
- Las presencias de z son cuatro, ligadas las tres primeras y libre la última.
- Finalmente las dos presencias de y son libres.

Definición 3 *Sea φ una fórmula. El conjunto de variables libres de φ, se denota $FV(\varphi)$.*
Es decir, $FV(\varphi) = \{x \in \mathsf{Var} \mid x \text{ figura libre en } \varphi\}$.
La notación $\varphi(x_1, \ldots, x_n)$ quiere decir que $\{x_1, \ldots, x_n\} \subseteq FV(\varphi)$.

Obsérvese que una misma variable x puede figurar tanto libre como ligada en una fórmula.

Definición 4 *Una fórmula φ es* cerrada *si no tiene variables libres, es decir, si $FV(\varphi) = \varnothing$. Una fórmula cerrada también se conoce como enunciado o sentencia.*

Definición 5 *Sea $\varphi(x_1, \ldots, x_n)$ una fórmula con $FV(\varphi) = \{x_1, \ldots, x_n\}$. La cerradura universal de φ, denotada $\forall\varphi$, es la fórmula $\forall x_1 \ldots \forall x_n\, \varphi$. La cerradura existencial de φ, denotada $\exists\varphi$ es la fórmula $\exists x_1 \ldots \exists x_n\, \varphi$.*

Obsérvese que una cerradura se obtiene cuantificando todas las variables libres de una fórmula.

1.1. Implementación

Se deja como ejercicio implementar todos los conceptos de esta sección mediante las siguientes funciones:

- Lista de variables libres: `fv :: Form -> [Nombre]`
- Lista de variables ligadas: `bv :: Form -> [Nombre]`
- Cerradura universal: `aCl :: Form -> Form`
- Cerradura existencial: `eCl :: Form -> Form`
- Alcances en una fórmula : `alcF :: Form -> [(Form,Form)]`, al tomar como entrada P esta función devuelve una lista de pares de la forma (A, B) donde A es una cuantificación que es subfórmula de P y B es el alcance de A.

2. Sustitución

La noción de sustitución en la lógica de predicados es más complicada que en la lógica proposicional debido a la existencia de términos y de variables ligadas. A diferencia de lo que sucede en lógica proposicional, donde una sustitución es sólo una operación textual sobre las fórmulas, en la lógica de predicados las sustituciones son operaciones sobre los términos y sobre las fórmulas y en este último caso deben respetar el ligado de variables, por lo tanto **no** son operaciones textuales.

Definición 6 *Una sustitución en un lenguaje de predicados $\mathcal{L}$ es una tupla de variables y términos denotada como $[x_1, x_2, \ldots, x_n := t_1, \ldots, x_n]$ donde*

- *$x_1, \ldots, x_n$ son variables distintas.*
- *$t_1, \ldots, t_n$ son términos de $\mathcal{L}$.*
- *$x_i \neq t_i$ para cada $1 \leq i \leq n$*

Por lo general denotaremos a una sustitución con $[\vec{x} := \vec{t}]$.

2.1. Implementación

Representaremos la sustitución $[\vec{x} := \vec{t}]$ mediante listas de pares $[(x_1, t_1), \ldots, (x_n, t_n)]$ Para esto definimos el tipo de sustituciones:

```
type Subst = [(Nombre,Term)]
```

Obsérvese que esto no garantiza que si `s:Subst` entonces `s` sea una sustitución legal. Por lo tanto debe escribirse una función `verifSus:: Subst -> Bool` que verifique si una lista dada es una sustitución.

2.2. Aplicación de una sustitución a un término

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 de define recursivamente como sigue:

$$\begin{aligned} x_i[\vec{x} := \vec{t}] &= t_i \qquad 1 \leq i \leq n \\ z[\vec{x} := \vec{t}] &= z \qquad \text{si } z \neq x_i \; 1 \leq i \leq n \\ c[\vec{x} := \vec{t}] &= c \qquad \text{si } c \in \mathcal{C}, \text{ es decir, } c \text{ constante} \\ f(t_1, \ldots, t_m)[\vec{x} := \vec{t}] &= f(t_1[\vec{x} := \vec{t}], \ldots, t_m[\vec{x} := \vec{t}]) \qquad \text{con } f^{(m)} \in \mathcal{F}. \end{aligned}$$

Obsérvese entonces que la aplicación de una sustitución a término es simplemente una sustitución textual tal y como sucede en lógica de proposiciones.

2.3. Implementación

La aplicación de una sustitución $[\vec{x} := \vec{t}]$ a un término t se implementa mediante una función

```
apsubT :: Term -> Subst -> Term
```

tal que si `s` implementa a la sustitución $[\vec{x} := \vec{t}]$ entonces `apsubT r s` debe devolver el término $r[\vec{x} := \vec{t}]$.

2.4. Sustitución en Fórmulas

La aplicación de sustituciones a fórmulas, necesita de ciertos cuidados debido a la presencia de variables ligadas mediante cuantificadores. La aplicación de una sustitución textual a una fórmula puede llevar a situaciones problemáticas con respecto a la sintaxis y a la semántica de la lógica. En particular deben evitarse los siguientes problemas:

Sustitución de variables ligadas La definición de sustitución textual

$$(\forall y \varphi)[\vec{x} := \vec{t}] =_{def} \forall (y[\vec{x} := \vec{t}])(\varphi[\vec{x} := \vec{t}])$$

obliga a sustituir todas las variables de una fórmula, pero tal definición genera expresiones que ni siquiera son fórmulas, por ejemplo tendríamos que

$$\big(\forall x P(y, f(x))\big)[x, y := g(y), z] = \forall g(y) P\big(z, f(g(y))\big)$$

La expresión de la derecha no es una fórmula puesto que la cuantificación sobre términos que no sean variables no está permitida. Lo mismo sucederá al definir la sustitución de esta manera para una fórmula existencial. En conclusión, la sustitución en fórmulas **NO** debe ser una sustitución textual puesto que las presencias ligadas de variables no pueden sustituirse.

Una solución inmediata al problema anterior consiste en definir la sustitución sólamente sobre variables libres, como sigue

$$(\forall x \varphi)[\vec{x} := \vec{t}] =_{def} \forall x (\varphi[\vec{x} := \vec{t}\,]_x)$$

donde $[\vec{x} := \vec{t}\,]_x$ denota a la sustitución que elimina al par $x := t$ de $[\vec{x} := \vec{t}]$.

Por ejemplo $[z, y, x := y, f(x), c]_x = [z, y := y, f(x)]$.
Y así la aplicación de sustitución para la fórmula del primer ejemplo se ejecuta como sigue:

$$\begin{array}{rcl}(\forall x P(y, f(x)))[x, y := g(y), z] & = & \forall x(P(y, f(x)[x, y := g(y), z]_x) \\ & = & \forall x(P(y, f(x))[y := z]) \\ & = & \forall x P(z, f(x))\end{array}$$

De esta manera se garantiza que el resultado de aplicar una sustitución a una cuantificación siempre será una fórmula. Análogamente podemos definir la sustitución para una fórmula existencial. Pasemos ahora a ver el segundo problema a evitar.

Captura de variables libres Si bien la solución dada al problema anterior es suficiente desde el punto de vista sintáctico, no lo es al entrar en juego la semántica o el significado de las fórmulas.

Considérese la fórmula $\forall x\varphi$, la semántica intuitiva nos dice que esta fórmula será cierta siempre y cuando $\varphi(x)$ sea cierta para cualquier valor posible de x, de manera que nos gustaría poder obtener, a partir de la de verdad $\forall x\varphi$, la verdad de cualquier aplicación de sustitución $\varphi[x := t]$.
Veamos ahora que que sucede si consideramos la fórmula

$$\forall x(\exists y(\, x \neq y))$$

Está fórmula expresa el hecho de que para cualquier individuo existe otro distinto de él y es cierta si el universo tiene al menos dos elementos.
Ahora bien, al momento de calcular el caso particular $(\exists y(x \neq y))[x := y]$ con la definición anterior de aplicación de sustituciones obtenemos:

$$(\exists y\,(x \neq y))[x := y] \;=\; \exists y(\,(x \neq y)[x := y]_y) \;=\; \exists y(\,(x \neq y)[x := y]) \;=\; \exists y\,(y \neq y)$$

Pero en esta última fórmula se ha cambiado el sentido: se está diciendo que hay un objeto y que es distinto de si mismo lo cual es absurdo.
De manera que con la definición dada no podemos garantizar la verdad de un caso particular de φ aun suponiendo la verdad de la cuantificación universal $\forall x\varphi$.

¿ Qué es lo que anda mal? obsérvese que la presencia libre de x en la fórmula $\exists y(x \neq y)$ se ligó al aplicar la sustitución $[x := y]$. Así que una variable libre que representa a un objeto particular se sustituyó por una variable que representa al objeto cuya existencia se está asegurando. Esta clase de sustituciones son inadmisibles, las posiciones que corresponden a una presencia libre de una variable representan a objetos particulares y el permitir que se liguen después de una sustitución modificará el significado intensional de la fórmula.

Existen dos maneras de solucionar el problema, la primera es aceptando la definición dada arriba mediante el uso de sustituciones de la forma $[\vec{x} := \vec{t}]_x$ pero prohibiendo la aplicación de sustituciones que liguen posiciones libres de variables. Para esto debe darse una una definición sustitución admisible para una fórmula, este método es el más usado en textos de lógica matemática.
Nosotros preferimos el segundo método, utilizado generalmente en la teoría de lenguajes de programación: *la aplicación de una sustitución a una fórmula se define renombrando variables ligadas de manera que siempre podremos obtener una sustitución admisible.*

Después de analizar los problemas anteriores, podemos dar una definición correcta para la sustitución:

Definición 7 *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

$$\begin{array}{rcl}
\bot[\vec{x} := \vec{t}] & = & \bot \\
\top[\vec{x} := \vec{t}] & = & \top \\
P(t_1, \ldots, t_m)[\vec{x} := \vec{t}] & = & P(t_1[\vec{x} := \vec{t}], \ldots, t_m[\vec{x} := \vec{t}]) \\
(t_1 = t_2)[\vec{x} := \vec{t}] & = & t_1[\vec{x} := \vec{t}] = t_2[\vec{x} := \vec{t}]
\end{array}$$

$$\begin{array}{rcl}
(\neg\varphi)[\vec{x} := \vec{t}] & = & \neg(\varphi[\vec{x} := \vec{t}]) \\
(\varphi \wedge \psi)[\vec{x} := \vec{t}] & = & (\varphi[\vec{x} := \vec{t}] \wedge \psi[\vec{x} := \vec{t}]) \\
(\varphi \vee \psi)[\vec{x} := \vec{t}] & = & (\varphi[\vec{x} := \vec{t}] \vee \psi[\vec{x} := \vec{t}]) \\
(\varphi \to \psi)[\vec{x} := \vec{t}] & = & (\varphi[\vec{x} := \vec{t}] \to \psi[\vec{x} := \vec{t}]) \\
(\varphi \leftrightarrow \psi)[\vec{x} := \vec{t}] & = & (\varphi[\vec{x} := \vec{t}] \leftrightarrow \psi[\vec{x} := \vec{t}])
\end{array}$$

$$\begin{array}{rcl}
(\forall y\varphi)[\vec{x} := \vec{t}] & = & \forall y(\varphi[\vec{x} := \vec{t}]) \ \ \textbf{si } y \notin \vec{x} \cup Var(\vec{t}) \\
(\exists y\varphi)[\vec{x} := \vec{t}] & = & \exists y(\varphi[\vec{x} := \vec{t}]) \ \ \textbf{si } y \notin \vec{x} \cup Var(\vec{t})
\end{array}$$

Mencionamos ahora algunas propiedades de la sustitución que pueden verificarse mediante inducción sobre las fórmulas.

Proposición 1 *La operación de sustitución tiene las siguientes propiedades, considerando $x \notin \vec{x}$.*

- *Si $x \notin FV(\varphi)$ entonces $\varphi[\vec{x}, x := \vec{t}, r\,] = \varphi[\vec{x} := \vec{t}]$.*
- *Si $\vec{x} \cap FV(\varphi) = \varnothing$ entonces $\varphi[\vec{x} := \vec{t}] = \varphi$.*
- *$FV(\varphi[\vec{x} := \vec{t}]) \subseteq (FV(\varphi) - \vec{x}) \cup Var(\vec{t})$.*
- *Si $x \notin \vec{x} \cup Var(\vec{t})$ entonces $x \in FV(\varphi)$ si y sólo si $x \in FV(\varphi[\vec{x} := \vec{t}])$.*
- *Si $x \notin \vec{x} \cup Var(\vec{t})$ entonces $\varphi[x := t][\vec{x} := \vec{t}] = \varphi[\vec{x} := \vec{t}\,][x := t[\vec{x} := \vec{t}]]$.*
- *Si $x \notin \vec{x} \cup Var(\vec{t})$ entonces $\varphi[\vec{x}, x := \vec{t}, t] = \varphi[\vec{x} := \vec{t}][x := t]$.*

2.5. Implementación

La aplicación de una sustitución $[\vec{x} := \vec{t}]$ a una fórmula A se implementa mediante una función

```
apsubF :: Form -> Subst -> Form
```

donde si `s` implementa a la sustitución $[\vec{x} := \vec{t}]$ entonces `apsubF A s` debe devolver la fórmula $A[\vec{x} := \vec{t}]$.

Hasta este momento debemos hacer la siguiente simplificación en la implementación:
en el caso de una sustitución donde la fórmula tiene como conectivo principal una cuantificación, primero se debe verificar la condición de las variables en la sustitución: *las variables ligadas **no** aparacen en ninguno de los términos de la sustitución.*
De no cumplirse convenimos en devolver la fórmula de entrada tal cual, por ejemplo el resultado de $(\forall x P(x, y))[y := f(x)]$ debe ser $\forall x P(x, y)$.

2.6. La relación de α-equivalencia

La definición de sustitución en fórmulas cuenta con una restricción aparente en el caso de los cuantificadores, por ejemplo, la sustitución

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

no está definida, puesto que x figura en $f(x)$ es decir $x \in Var(f(x))$, con lo que no se cumple la condición necesaria para aplicar la sustitución, a saber que las variables en la sustitución son ajenas a las de la fórmula. Por lo tanto, la aplicación de cualquier sustitución a una fórmula hasta ahora es una función parcial.

Esta aparente restricción desaparece al notar que los nombres de las variables ligadas no importan: por ejemplo, las fórmulas $\forall x P(x)$ y $\forall y P(y)$ significan exactamente lo mismo, a saber que todos los elementos del universo dado cumplen la propiedad P.

Por lo tanto convenimos en identificar fórmulas que sólo difieren en sus variables ligadas, esto se hace formalmente mediante la llamada relación de α-equivalencia definida como sigue:

Definición 8 *Decimos que dos fórmulas φ_1, φ_2 son α-equivalentes lo cual escribimos $\varphi_1 \sim_\alpha \varphi_2$ si y sólo si φ_1 y φ_2 difieren a lo más en los nombres de sus variables ligadas.*

Ejemplo 2.1 Las siguientes expresiones son α-equivalentes.

$$\forall x P(x,y) \to \exists y R(x,y,z) \quad \sim_\alpha \quad \forall w P(w,y) \to \exists v R(x,v,z) \quad \sim_\alpha \quad \forall z P(z,y) \to \exists u R(x,u,z)$$

Más tarde demostraremos que las fórmulas α-equivalentes también son lógicamente equivalentes y por lo tanto son intercambiables en cualquier contexto o bajo cualquier operación.

Usando la α-equivalencia, la operación de sustitución en fórmulas se vuelve una función *total* por lo que siempre está definida.

Ejemplo 2.2 Por ejemplo se tiene que

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

Veamos otros ejemplos de sustituciones:

Ejemplo 2.3 Las siguientes sustituciones se sirven, en caso necesario, de la α-equivalencia

$$\begin{aligned}
\forall x R(x,y,f(z))[y,z := d, g(w)] &= \forall x R(x,d,f(g(w))) \\
\forall x P(x,y)[y,z := f(c), w] &= \forall x P(x,f(c)) \\
Q(y,z,y)[y,z := g(d), f(y)] &= Q(g(d), f(y), g(d)) \\
\exists w Q(y,z,y)[y,z := g(d), f(y)] &= \exists w Q(g(d), f(y), g(d)) \\
\exists y Q(y,z,y)[y,z := g(d), f(y)] &= \exists w Q(w, f(y), w) \\
\exists z Q(y,z,y)[y,z := g(d), f(y)] &= \exists w Q(g(d), w, g(d)) \\
\forall x(Q(z,y,x) \wedge \exists z T(f(z),w,y))[x,y,z := a, z, g(w)] &= \forall u(Q(g(w),z,u) \wedge \exists v T(f(v),w,z)) \\
\forall x(Q(z,y,x) \wedge \exists z T(f(z),w,y))[z,w,y := g(x), h(z), w] &= \forall u(Q(g(x),w,u) \wedge \exists v T(f(v),h(z),w))
\end{aligned}$$

2.7. Implementación

Se deja como ejercicio implementar un test para la α-equivalencia así como redefinir la función de sustitución usando el renombre de variables ligadas. Estos ejercicios son más complicados que los anteriores.

- `vAlfaEq:: Form -> Form -> Bool` verifica si dos fórmulas son α-equivalentes.
- `renVL:: Form -> Form` renombra las variables ligadas de una fórmula de manera que las listas de variables libres y ligadas sean ajenas. Esto es un caso particular de la siguiente función.
- `renVLconj :: Form -> [Nombre] -> Form` renombra las variables ligadas de una fórmula de forma que sus nombres sean ajenos a los de una lista dada.
- `apSubF2 :: Form -> Subst -> Form` que implemente la sustitución en fórmulas usando la α-equivalencia.